UNIVERSIDADE PRESBITERIANA MACKENZIE
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
Trabalho de Graduação Interdisciplinar

ANÍSIO RODRIGUES NETO
MARCOS ALVES PEREIRA
RODRIGO DE OLIVEIRA NEVES

GERENCIADOR DE PROJETOS WEB (JEE)
DESENVOLVIMENTO ORIENTADO A OBJETOS USANDO METODOLOGIA RUP
UM ESTUDO DE CASO

São Paulo
2007

ANÍSIO RODRIGUES NETO
MARCOS ALVES PEREIRA
RODRIGO DE OLIVEIRA NEVES

GERENCIADOR DE PROJETOS WEB (JEE)
DESENVOLVIMENTO ORIENTADO A OBJETOS USANDO METODOLOGIA RUP
UM ESTUDO DE CASO

Trabalho de conclusão de Curso de Sistemas de Informação da Universidade Presbiteriana Mackenzie, apresentado como requisito parcial para a obtenção do Grau de Bacharel em Sistemas de Informação.

ORIENTADOR: Prof. MSc Vinicius Miana Bezerra

São Paulo
2007

UNIVERSIDADE PRESBITERIANA MACKENZIE
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
SISTEMAS DE INFORMAÇÃO
Termo de Julgamento de Defesa de Trabalho de Graduação Interdisciplinar

Aos ____ dias do mês de junho de 2007, às ______ horas, no prédio ___, sala _____ da Universidade Presbiteriana Mackenzie, presente a Comissão Julgadora, integrada pelos senhores Professores, abaixo discriminados, iniciou-se a apresentação do Trabalho de Graduação Interdisciplinar do Grupo de Trabalho formado pelos alunos abaixo e concluída a argüição, procedeu-se ao julgamento na forma regulamentar, tendo a Comissão Julgadora atribuído as seguintes notas aos Candidatos

Título do Trabalho: Gerenciador de Projetos Web (JEE) – Desenvolvimento Orientado a Objetos usando Metodologia RUP – Um Estudo de Caso

ALUNOS	Número Matrícula	Média da Banca	Nota Orientador	Média	Desconto por atraso	Nota Final
1. Anisio Rodrigues Neto	403.2684-5					
2. Marcos Alves Pereira	403.4974-8					
3. Rodrigo de Oliveira Neves	403.2569-5					

COMISSÃO JULGADORA		NOTA
Orientador		
Titular 1		
Titular 2		
Suplente		
Média Obtida pelo Grupo de Trabalho		

Para constar, é lavrado o presente termo que vai assinado pela Comissão Julgadora e pelo Coordenador de TGI.

São Paulo,.

Comissão Julgadora

____
Prof.

____
Prof.

____
Prof.

Coordenador de TGI

____
Prof.

Às nossas famílias, amigos e professores.

AGRADECIMENTOS

À Deus, que esteve sempre presente em todos os momentos, dando-nos coragem e determinação.

Às nossas famílias, pelo incentivo e apoio nos momentos difíceis.

Ao professor Vinicius Miana Bezerra, por nos orientar e encorajar na realização deste trabalho.

À todos os colegas do curso de Sistemas de Informação, a nossa gratidão pela ajuda e colaboração ao longo destes anos.

RESUMO

O gerenciamento de projetos se tornou fator crítico de sucesso com relação à competitividade entre as empresas. Na área de desenvolvimento de sistemas, metodologias são adotadas para minimizar falhas ao longo do ciclo de vida do projeto. Em alguns casos, a escolha de uma determinada metodologia, devido à sua complexidade, pode dificultar seu andamento. Cada projeto possui características próprias e a aplicação de uma mesma metodologia nem sempre resultará em eficiência. Cabe à empresa optar e adequar cada metodologia a seus diferentes projetos. O objetivo deste trabalho é viabilizar a implementação de um gerenciador de projetos, baseando-se em uma das mais utilizadas metodologias de desenvolvimento de sistemas, a *Rational Unified Process* (RUP). A arquitetura de desenvolvimento escolhida foi o *Java Enterprise Edition* (JEE), pois atende as necessidades do projeto, trabalhar com orientação a objetos e faz *interface* com a plataforma *web*.

Palavras-chave: RUP. Gerenciamento de projetos. UML. JEE. Metodologia de desenvolvimento. Orientação a objetos.

ABSTRACT

Project management became a critical success factor about competitiveness between companies. In system deployment area methodologies are adopted do reduce failures along the project life-time. But some times the choice of a particular methodology, due to its complexity, can disturb its progress. Each project has its own properties and the use of the same methodology not always will work with the same efficiency. The company has to choose and adapt each methodology to its different projects. This paper´s goal is the implementation of a project manager system, using one of the most used system development methodologies, the Rational Unified Process (RUP). The chosen development architecture was the Java Enterprise Edition (JEE), because it suits the project needs, work with object orientation and build an web interface.

Keywords: RUP. Project management. UML. JEE. Development methodology. Object orientation.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

Cap. I – Introdução

1.1. Justificativa do Tema

As organizações constantemente executam projetos. Seja no desenvolvimento de um novo produto ou serviço, na implantação de um novo sistema gerencial ou até mesmo em mudanças organizacionais.

Os profissionais da área de gerenciamento de projetos tem enfrentado grande dificuldade em controlar e administrar estes projetos. O fracasso começa a se tornar algo muito freqüente e soluções para que estas estatísticas mudem têm sido incorporadas às empresas. (Vargas, 2006).

Segundo Prado (2003, p.19), a obtenção de sucesso em projetos se tornou algo muito importante para as empresas, pois através disto, ela pode se tornar mais competitiva no mercado e seus objetivos são rapidamente alcançados, e também, com mais facilidade. Na área de desenvolvimento de software não é diferente. Um software nada mais é do que o resultado de um projeto, portanto enfrenta os mesmos problemas, podendo também, ter como seu futuro, o fracasso.

Metodologias de gerenciamento de projetos estão sendo implantadas nas empresas hoje em dia, com o objetivo de proporcionar uma melhor administração e controle dos sistemas que estão sendo desenvolvidos, desde sua documentação até a fase de implementação. Cada projeto, possui características únicas e, esse fato, dificulta a adoção de uma metodologia única para todos os projetos. Para contornar esse tipo de problema, em muitos casos, são elaboradas novas metodologia com intuito de atender todos os diferentes tipos de projetos. Essa elaboração, na maioria das vezes, é feita com base em metodologias já existentes e consagradas. (Prado, 2003).

Muitas metodologias são complexas, e não servem para projetos de pequeno porte. E de contraponto, existem metodologias mais simples, que não atendem projetos com grande porte.

Para o desenvolvimento de sistemas orientados a objetos, a *Rational Unified Process* (RUP) possui características que a colocam como uma das principais metodologias adotadas nas organizações. Porém, em cada etapa do ciclo de vida do projeto, muitos documentos são gerados, o que pode ocasionar atrasos na sua entrega se for de baixa complexidade. Para evitar que ocorram esses atrasos, e também, continuar utilizando a mesma metodologia em projetos de níveis de complexidade distintos, faz-se necessário a escolha de apenas alguns dos diversos documentos que podem ser gerados ao longo do ciclo de vida do projeto. (Prado, 2003), (Vargas, 2006).

E, devido à importância deste tema, o RUP foi adotado para o desenvolvimento deste trabalho.

1.2. Objetivos

Este trabalho tem por objetivo mostrar as facilidades e dificuldades são encontradas no processo de desenvolvimento de sistemas com a utilização da metodologia RUP. Para este estudo, será implementado um gerenciador de projetos web na plataforma *Java Enterprise Edition* (JEE), utilizando alguns conceitos básicos de gerenciamento de projetos abordados neste trabalho.

1.3. Procedimentos metodológicos

O procedimento metodológico escolhido foi a elaboração de um estudo de caso. Este estudo compreende a implementação de um sistema de gerenciamento de projetos utilizando com base de desenvolvimento, a orientação a objetos. Como metodologia de desenvolvimento será utilizado o RUP e, em cada fase do ciclo de vida do projeto, serão gerados os documentos que forem julgados necessários.

1.4. Organização do Trabalho

No capítulo 2, "Gerenciamento de Projetos", será feita uma breve introdução sobre o título em questão, informando sua importância, os problemas encontrados e as formas de gerenciamento.

No capítulo 3, uma breve introdução às características e à utilização da *Unified Modeling Language* (UML) será apresentada pelo fato de o RUP utilizá-la para gerar os artefatos.

Serão estudadas as quatro fases do RUP, concepção, elaboração, construção e transição, no capítulo 4, como os artefatos são gerados ao longo de seu ciclo de vida que compreende desde modelos, códigos-fontes até diagramas. Todos estes documentos são de extrema importância para o controle no processo de desenvolvimento de software.

No capítulo 5, será abordado o projeto em questão, discutindo sobre quais artefatos foram escolhidos e como foi o processo de elaboração e execução, desde seu início até sua conclusão. Serão expostas também, as dificuldades encontradas ao longo do desenvolvimento e, os motivos das escolhas de determinados caminhos que o projeto seguiu, entre eles artefatos e tecnologias utilizadas.

E por fim, no capítulo 6, "Conclusão", serão apresentados os resultados obtidos através do desenvolvimento do estudo de caso aqui descrito.

Cap. II – Gerenciamento de Projetos

> “Executar projetos ou executar empreendimentos é uma característica de sobrevivência da empresa moderna.” (Prado, 2003).

Nos tempos atuais, a competitividade existente entre as empresas para a conquista de clientes e, a dificuldade na entrega de produtos ou serviços tem crescido. Prado (2003), afirma que isso se deve ao fato de os projetos serem cada vez mais complexos e com prazos mais curtos para seu desenvolvimento. O gerenciamento de projetos possibilita através de diversas metodologias, o cumprimento destes prazos, alcance de metas e a conquista de credibilidade junto ao cliente.

Os projetos existem em todos os momentos, desde o pessoal até o profissional, na organização de uma festa, no lançamento de um novo produto ou no desenvolvimento de um novo sistema. (Prado, 2003)

Prado (2003, p.57) afirma que um projeto pode ser definido como bem sucedido quando:

- Atingiu as metas estabelecidas;
- Atendeu às expectativas dos envolvidos no projeto;
- Obteve mínimas mudanças em seu escopo;
- Não violentou valores ou cultura da organização;
- Não violentou o fluxo usual da organização.

Segundo Prado (2003, p.19), o gerenciamento de projetos surgiu na década de setenta e desde então tem evoluído constantemente, estando presente hoje em dia em quase todos os ramos de atividade, como mecanismo fundamental para atingir o sucesso.

Uma grande dificuldade encontrada em gerenciamento de projetos é fazer com que este não atinja o fracasso.

> “Pesquisas têm comprovado que a ocorrência de fracassos é muito comum; em algumas áreas de negócios os números chegam a assustar. Por exemplo, na indústria de desenvolvimento de software, pesquisas da Gartner Group apontam que apenas 20% dos projetos são bem sucedidos.” (Prado, 2003).

Vargas (2006, p44) diz que:

> “Projeto problemático pode ser definido como um projeto cuja variação entre o esperado e o realizado excedeu os limites de tolerância aceitáveis. [...]”

Às vezes definir o “não-sucesso” de um projeto como fracasso é injusto até mesmo para com o gerente em questão, pois fatores externos e internos podem interferir na sua entrega, como por exemplo um concorrente lançar um produto com melhor competitividade de mercado. Neste momento o projeto em questão não falhou e sim teve um desvio de meta.

Vargas (2006, p44), apresenta diversos fatores que podem identificar um “projeto-problema”. Esses indicadores podem estar relacionados com os *stakeholders**, os recursos, a documentação, o escopo, os ricos, os custos e s prazos.

Veja na figura abaixo o mapa mental dos indicadores de um projeto problemático. (Vargas, 2006, p45).

**Stakeholders* – Todos os envolvidos no projeto. Como por exemplo, gerentes, consultores, clientes, analistas e programadores.

Figura 2.1 – Mapa mental dos indicadores de um projeto problemático.

Fonte: Vargas (2006, p45).

Todos os fatores citados acima, segundo Ricardo Vargas (2006, p44), não indicam, se analisados isoladamente, se um projeto possui algum tipo de problema. Todos devem ser analisados para que a identificação de um problema existente seja realmente válida e concisa.

Entende-se que para que um projeto alcance o seu principal objetivo, o sucesso, fatores internos e externos da organização devem ser levados em consideração, assim como também a utilização de uma metodologia e ferramentas que possibilitam um melhor gerenciamento.

Prado (2003), define metodologia como:

> [...] conjunto de técnicas, regras e métodos orientados para um fim em comum [...]

Sendo assim, pode-se dizer que uma metodologia é o mecanismo utilizado para definir um padrão a ser seguido por todos os *stakeholders* e é muito necessária para que um projeto possua maiores chances de atingir seu principal objetivo: o sucesso.

A metodologia a ser utilizada no gerenciamento de projetos pode ser desenvolvida pela organização ou simplesmente pode-se adotar metodologias já existentes no mercado.

Segundo Prado (2003, p.140), uma metodologia de gerenciamento de projetos deve contemplar os processos de gerenciamento e as áreas de atuação gerencial, ou seja, deve-se utilizar uma metodologia que possibilite que o gerente de projeto tenha o maior controle possível sobre este, desde o início até a conclusão.

2.1. PMBOK

O *Project Management Body of Knowledge* (PMBOK) do *Project Management Institute* (PMI) não é uma metodologia, e sim um guia do conjunto de conhecimentos em gerenciamento de projetos.

O guia PMBOK identifica os pontos pertinentes que são aplicáveis na maioria dos projetos em boa parte do tempo. Apresenta uma visão geral e não algo com muitos detalhamentos.

Mostra que com a aplicação correta de habilidades, ferramentas e técnicas, as chances de um projeto atingir o sucesso são grandes.

A estrutura do guia PMBOK é organizada em três seções (PMBOK, 2004):

- A primeira seção, “A estrutura do gerenciamento de projetos”, fornece uma estrutura básica para que o leitor entenda sobre gerenciamento de projetos. Nesta seção há o capítulo de Introdução, onde é fornecido uma visão geral do restante do guia e o capítulo de Ciclo de Vida e Organização do Projeto, que descreve o ambiente no qual o projeto opera;
- Na segunda seção, “A norma de gerenciamento de projetos de um projeto” aborda os cinco grupos de processos de gerenciamento de projetos necessários para qualquer projeto e os processos de gerenciamento de projetos que os compõem;
- A terceira seção, aborda as áreas de conhecimento em gerenciamento de projetos, estando elas divididas cada uma em um capítulo (PMBOK, 2004):

    - Gerenciamento de integração do projeto;
    - Gerenciamento do escopo do projeto;
    - Gerenciamento de tempo do projeto;
    - Gerenciamento de custos do projeto;
    - Gerenciamento da qualidade do projeto;
    - Gerenciamento de recursos humanos do projeto;
    - Gerenciamento das comunicações do projeto;
    - Gerenciamento de riscos do projeto;
    - Gerenciamento de aquisições do projeto.

Conforme descrito no PMBOK (2004), Gerenciamento de Integração de Projeto assegura que os processos e as atividades que integram os diversos elementos do projeto estejam adequadamente coordenados.

Os principais processos que correspondem ao Gerenciamento de Integração de Projeto são (PMBOK, 2004):

- Desenvolver o termo de abertura do projeto, ou seja, uma autorização formal do projeto;
- Desenvolver a declaração do escopo preliminar, onde possibilita uma visão de alto nível do escopo do projeto;
- Desenvolvimento do plano de gerenciamento do projeto, onde faz-se a construção de um documento coerente e defini-se como o projeto é executado, monitorado, controlado e encerrado;

- Orientar e gerenciar a execução do projeto, que consiste em realizar várias ações para a realização do trabalho que foi definido no escopo do projeto;
- Monitorar e controlar o trabalho do projeto, onde é feita uma verificação do andamento, desempenho e avaliações para efetuar melhorias no processo. Um monitoramento contínuo permite que a equipe identifique as áreas que exigem uma atenção especial.
- Controle integrado de mudanças, que é realizado desde o início do projeto até o seu término. Permite o controle das mudanças que ocorreram durante todo o projeto.
- Encerrar o projeto, onde vários processos são envolvidos, tais como: encerramento do processo administrativo, de contrato, e o de formalização da entrega do projeto.

Segundo o PMBOK (2004), Gerenciamento de Escopo do Projeto é composto dos "processos necessários para garantir que o projeto inclua todo o trabalho necessário, e somente ele, para terminar o projeto com sucesso".

Os principais processos que correspondem ao Gerenciamento de Escopo de Projeto são (PMBOK, 2004):

- Planejamento do escopo, onde se cria um plano de gerenciamento de escopo do projeto que documente como o escopo será definido;
- Definição do escopo, onde se faz sua declaração detalhadamente;
- Criar Estrutura Analítica do Projeto (EAP) onde é feita uma subdivisão das principais entregas do projeto em componentes menores e mais fáceis de serem gerenciados;
- Verificação do escopo, que é o processo de aceitar formalmente o trabalho do projeto, conforme já definido em sua documentação, no escopo ou no contrato.
- Controle do escopo, onde é feito o controle das mudanças no escopo do projeto, garantindo que mudanças sejam acordadas por todos e que possa ser determinado quando alguma mudança ocorreu.

O guia PMBOK (2004), através do Gerenciamento de Tempo de Projeto, define os processos necessários, para garantir que, o projeto cumpra os prazos definidos em um cronograma de atividades, e, seja entregue dentro do prazo estipulado.

Os principais processos que correspondem ao Gerenciamento de Tempo de Projeto são (PMBOK, 2004):

- Definições da atividade, onde são feitas identificações das atividades específicas do cronograma que necessitem ser executadas;
- Seqüênciamento de atividades, que consiste na identificação e documentação das dependências das atividades existentes no cronograma;
- Estimativa de recursos da atividade, onde se faz estimativa dos recursos que serão necessários para executar cada atividade do cronograma;

- Estimativa de duração da atividade, onde se estima o período necessário para a conclusão de cada atividade do cronograma;
- Desenvolvimento do cronograma, onde é criado o cronograma do projeto;
- Controle do cronograma, onde é feito um controle das alterações efetuadas no cronograma.

Conforme o PMBOK (2004), Gerenciamento de Custo do Projeto define os processos necessários para certificar que o projeto será concluído dentro do orçamento previsto que já foi aprovado.

Os principais processos que correspondem ao Gerenciamento de Custo de Projeto são (PMBOK, 2004):

- Estimativa de custos, onde se desenvolve uma estimativa dos custos dos recursos que serão necessários para terminar cada atividade do projeto;
- Orçamentação, onde há uma agregação dos custos que foram estimados para estabelecer uma linha de base de custos;
- Controle de custos, onde faz-se o controle das mudanças no orçamento do projeto e dos fatores que criam as variações de custos.

O Gerenciamento de Tempo do Projeto e o Gerenciamento de Custo do Projeto são as áreas de conhecimento de maior importância dentro do contexto do gerenciamento de um projeto, pois são as mais visíveis durante a sua gestão.

O Gerenciamento de Qualidade de Projetos, é definido pelo PMBOK (2004) como sendo a área cujo principal objetivo é garantir que o projeto será concluído dentro da qualidade esperada, garantindo desta forma a satisfação de todos os envolvidos no projeto, os *stakeholders*.

Os principais processos que correspondem ao Gerenciamento de Qualidade de Projeto são (PMBOK, 2004):

- Planejamento da qualidade, onde os padrões de qualidade que são relevantes são identificados e é elaborada uma solução de como satisfazê-los;
- Realizar a garantia da qualidade, que é o processo onde as qualidades que foram planejadas são aplicadas nas atividades para garantir que o projeto empregue todos os processos necessários para que atendam os requisitos;
- Realizar o controle de qualidade, que consiste no monitoramento para certificar que os padrões relevantes de qualidade estão sendo levados em consideração, eliminando assim um desempenho que não seja satisfatório.

Segundo o PMBOK (2004), Gerenciamento de Recursos Humanos do Projeto define os processo que organizam e gerenciam toda a equipe do projeto.

Paul Campbell Dinsmore e Fernando H. Da Silveira Neto (2006) afirmam que:

> "Se as pessoas falham em agir, acompanhar, tomar decisões, analisar ou avaliar, os projetos desviam-se do seu curso."

No fator humano pode-se concentrar pelo menos metade dos problemas em projetos. Ao dar-se atenção a esse fator no gerenciamento de projetos, pode-se produzir-lhe resultados positivos. Paul Campbell Dinsmore e Fernando H. Da Silveira Neto (2006), definem algumas formas de resultado positivo, tais como: geração de sinergia, formação de contratos psicológicos, criação de um arranjo produtivo, eliminação de dificuldades organizacionais, melhoria nas relações com o cliente e gerenciamento de projetos mais eficaz.

Os principais processos que correspondem ao Gerenciamento de Recursos Humanos do Projeto são (PMBOK, 2004):

- Planejamento de recursos humanos, onde se identifica e documenta as funções, responsabilidades e relações hierárquicas do projeto;
- Contratar ou mobilizar a equipe do projeto, que consiste na obtenção do pessoal necessário para finalizar-lo;
- Desenvolver a equipe do projeto, que preza a melhoria de competências e interação de membros da equipe para aprimorar o desempenho do projeto;
- Gerenciar a equipe do projeto, onde acompanha-se o desempenho dos membros da equipe buscando sempre melhorar o desempenho do projeto.

O Gerenciamento das Comunicações do Projeto, conforme definido pelo PMBOK (2004), determina os processos necessários para que as informações sobre o projeto possam ser manipuladas de forma oportuna e adequada. Todos os *stakeholders* devem entender como as comunicações afetam o projeto como um todo.

Os principais processos que correspondem ao Gerenciamento das Comunicações do Projeto são (PMBOK, 2004):

- Planejamento das comunicações, onde são determinadas as necessidades de informações e comunicações dos *stakeholders*;
- Distribuição das informações, onde as informações necessárias são colocadas à disposição dos *stakeholders*;
- Relatório de desempenho, que consiste na coleta e distribuição das informações sobre o desempenho, incluindo o relatório de andamento e medição do progresso e previsão;
- Gerenciar as partes interessadas, onde gerenciam-se as comunicações para satisfazer os *stakeholders* e resolver seus problemas.

Já o Gerenciamento de Riscos do Projeto, segundo o PMBOK (2004), determina os processos que tratam da realização de identificação, análise, repostas, monitoramento, controle e planejamento do gerenciamento de riscos em um projeto e, na maioria das vezes, esses processos sofrem atualizações durante todo seu ciclo de vida.

Como principal objetivo, o gerenciamento de riscos do projeto procura aumentar a probabilidade de eventos positivos e diminuir os negativos.

Os principais processos que correspondem ao Gerenciamento de Risco do Projeto são (PMBOK, 2004):

- Planejamento do gerenciamento de risco, onde existe a decisão de como deverá ser abordada, planejada e executada as atividades de gerenciamento de riscos de um projeto;
- Identificação de riscos, onde tenta-se identificar os riscos que podem afetar o projeto e documentar suas características;
- Análise qualitativa de riscos, onde coloca-se prioridade para cada risco e sua análise;
- Análise quantitativa de riscos, que é a análise numérica dos efeitos dos riscos que foram identificados;
- Planejamento de respostas a riscos, onde desenvolvem-se opções e ações para que possam aumentar oportunidades e reduzir ameaças aos objetivos do projeto;
- Monitoramento e controle de riscos, que é o acompanhamento dos riscos durante todo o ciclo de vida do projeto.

A última das áreas de conhecimento é o Gerenciamento de Aquisições de Projetos. Este gerenciamento, segundo o PMBOK (2004), define os processos que são necessários para comprar ou adquirir os produtos, serviços ou resultados necessários de fora da equipe do projeto para a realização do trabalho. Também engloba os processos de gerenciamento de contratos.

Os principais processos que correspondem ao Gerenciamento de Aquisições do Projeto são (PMBOK, 2004):

- Planejar compras e aquisições, onde determina-se o que comprar ou adquirir e como e quando fazer isso;
- Planejar contratações, onde faz-se a documentação dos requisitos de produtos, serviços e resultados e identificação de possíveis fornecedores;
- Solicitar respostas de fornecedores, onde obtêm-se informações, cotações, preços, quando necessário;
- Selecionar fornecedores, onde analisa-se as ofertas e escolhe-se possíveis fornecedores, negociando um contrato por escrito;
- Administração de contrato, que é a parte de gerenciamento do contrato entre o comprador e o fornecedor;
- Encerramento do contrato, onde cada contrato é terminado ou liquidado.

Pode-se ter uma visão geral das áreas de conhecimento em gerenciamento de projetos e processos através da figura abaixo:

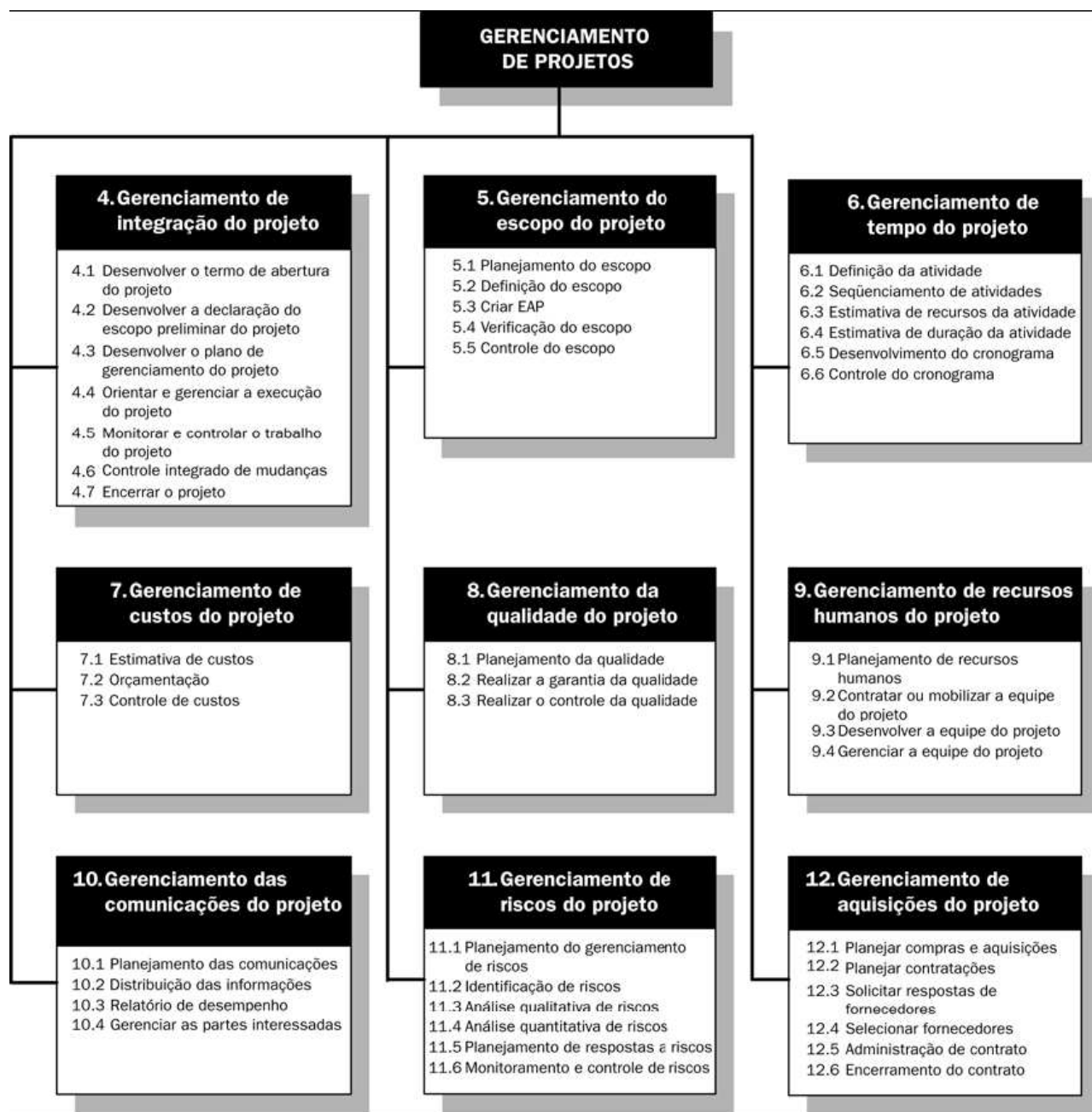



Figura 2.2 – Visão geral das áreas de conhecimento em gerenciamento de projetos e os processos de gerenciamento de projetos.

Fonte: Guia PMBOK (PMBOK, 2004).

2.2. Ferramentas de gerenciamento de projetos

Outro fator importante, além da metodologia utilizada é o uso de Sistemas de Gerenciamento de Projetos (SGP), que possibilitam ao gerente de projetos diversas formas de gerenciá-lo, entre elas:

- Maior controle das atividades dos *stakeholders*;
- Maior controle de prazos;
- Verificar se o projeto não está fugindo do seu escopo.

Segundo Prado (2003), um SGP deve refletir a metodologia da organização e na maioria das vezes a compra de um SGP completo nem sempre é possível. Ou seja, a maioria das empresas adota metodologias diferentes para o gerenciamento de seus projetos, logo, para fazer um bom gerenciamento em algumas ocasiões, torna-se necessário o desenvolvimento de um sistema específico que atenda às necessidades da empresa.

No mercado atualmente existem diversos SGP. Dentre eles encontra-se desde soluções de Softwares Livres, onde muitas vezes não existe custo, até soluções proprietárias com alto custo de licença.

Existem sistemas tanto com interface *web* quanto para instalação *desktop*. Como exemplo de SGP com interface *web* pode-se citar: dotProject, ProjectOpen e Gforce. E para instalação *desktop*: Gantt Project, MSProject, Planner e Real Time Project.

O dotProject é um sistema totalmente baseado em *web*. Foi desenvolvido em PHP e seus criadores indicam a utilização da combinação clássica LAMP (Linux, Apache, MySQL e PHP). Possui instalação simples para usuários que estão familiarizados com a configuração de servidores e, por seu banco de dados ser MySQL, permite fácil manipulação dos dados (DOTPROJECT, 2007).

O ProjectOpen tem como objetivo principal a administração dos custos de um projeto e a colaboração dos *stakeholders*. Possui configurações próprias para diferentes ambientes de uso. Por exemplo: uma empresa de consultoria usaria uma configuração diferente de uma empresa de publicidade. De todos os SGP, o ProjectOpen possui melhor segurança e integração, dividindo de forma muito eficiente o acesso dos *stakeholders* com funções distintas. Duas grandes desvantagens para o mercado brasileiro se destacam, o software não possui versão em português e a linguagem utilizada é a TCL (linguagem pouco utilizada nos tempos atuais). (PROJECT-OPEN, 2007).

O Gforge atende a todas as necessidades do ciclo de vida de um projeto. Atua também como um sistema de registro de *bugs* e de repositório de distribuição (códigos fontes, executáveis). Possui um *plugin* comercial que é capaz de interagir com o MSProject da Microsoft. Foi desenvolvido em PHP, Linux, Apache e banco de dados PostgreSQL, e outros componentes opcionais. Hoje em dia existe uma versão comercial disponível pelo Gforce Group, chamada Enterprise CDE. (GFORCE, 2007).

O Gantt Project é focado na elaboração de cronogramas para necessidades simples. Foi desenvolvido em Java podendo ser utilizado em vários sistemas operacionais. Na versão 2, seus

dados podem ser importados e exportados no formato do MSProject. Bastante indicado em projetos onde o cronograma é de extrema importância. (GANTTPROJECT, 2007).

O Planner nada mais é que a continuação do Mr Project, software que já foi bastante popular entre usuários de Software Livre. O programa faz parte do Gnome Office, estando assim incluído em diversas distribuições Linux. As características e interface lembram muito o MSProject. O banco de dados utilizado é o PostgreSQL. É recomendado para usuários que trabalham com estações Linux possuem conhecimentos de informativas mais avançados e necessitam calcular o caminho crítico de um projeto. (IMENDIO, 2007).

O Real Time Project tem como pontos fortes o cronograma, a estrutura analítica do projeto, o controle de recursos e o controle de custos. Está disponível para Unix/Linux, Windows e Macintosh, em forma de binários específicos. Sua interface é bastante amigável e é o competidor direto do MSProject. É recomendado para empresas ou usuários que necessitem de uma solução abrangente de gerenciamento de projetos. (AMS, 2007).

E por fim, pode-se citar o MSProject da Microsoft. Não diferente de muitas outras soluções Microsoft, o MSProject não é *Open Source*. Pode-se importar dados de planilhas e bases de dados transformando-os em informações de folhas de projetos. Possibilita a integração com as demais ferramentas *Office* da Microsoft, tais como: Outlook, Excel e Word. É um dos softwares de gerenciamento de projetos mais utilizados e vendidos no mercado. (MICROSOFT, 2007).

> No mundo da web 2.0 pouco importa onde está o usuário: num PC, num celular ou em qualquer outro dispositivo que entenda o significado da palavra rede. O software, por sua vez, pode estar só na internet ou ser uma combinação entre o que fica no computador e na rede. A tendência é virar algo cada vez mais 100% online. (Fortes, 2006, p.48).

Nos tempos atuais, a tendência é o crescimento da utilização de sistemas *web*, tanto pela mobilidade e independência de sistema operacional quanto pelo baixo custo.

2.3. Comparativo de sistemas de gerenciamento de projetos

A maioria dos sistemas de gerenciamento de projetos tem as mesmas características e funcionalidades. Algumas diferenças podem ser encontradas no que se refere a relatórios que fornecem uma análise mais detalhado do projeto que está sendo controlado.

Abaixo, segue uma tabela de comparação dos sistemas de gerenciamento de projetos citados no item anterior:

SGP	Forma de Interação	Principais Características	Plataformas
dotProject	Interface Web	Classificação de projetos, gerenciamento de documentação, emissão de alertas sobre alterações nos documentos.	Unix Linux Windows
ProjectOpen	Interface Web	Administração dos custos de um projeto e a colaboração entre membros da equipe.	Linux Windows
Gforce	Interface Web	Atende às necessidades de todo o ciclo de vida do projeto, controle de registro de *bugs* e de repositório de distribuição.	Linux Windows

Gantt Project	Instalação *Desktop*	Elaboração de cronogramas.	Unix Linux Windows Macintosh
Planner	Instalação *Desktop*	Recursos de cronogramas relativamente avançados.	Linux Windows
Real Time Project	Instalação *Desktop*	Controle de cronograma, estrutura analítica do projeto, custos e recursos	Unix Linux Windows Macintosh
MSProject	Instalação *Desktop*	Importação de dados de planilhas e bases de dados transformando-os em informações de folhas de projetos e a integração com as demais ferramentas da Microsoft Office.	Windows

Tabela 1.1. – Comparativo de sistemas de gerenciamento de projetos.

CAP. III – UML – Unified Modeling Language

Ivar Jacobson, James Rumbough e Grady Booch são os fundadores da *Unified Modeling Language*, ou UML, como é conhecida. Esta linguagem é utilizada como padrão de documentação de projetos de *software.* (Sommerville, 2005).

Antes da junção para a criação da UML, cada um deles estava desenvolvendo um método de documentação diferente. Ivar Jacobson, da empresa Objectory desenvolvia o OOSE, Grady Booch da empresa Rational Software Corporation, o Booch e por fim, James Rumbough, da empresa General Eletrics, o OMT. Jacobson, Rumbaugh e Booch (2005).

Jacobson, Rumbaugh e Booch (2005) iniciaram a criação da linguagem por três motivos: o primeiro, porque os métodos já estavam evoluindo todos para uma mesma direção. O segundo, porque a unificação dos métodos traria alguma estabilidade ao mercado orientado a objetos. E, o terceiro, o aprimoramento dos três métodos anteriores de modo que poderiam lidar com problemas que antes não se poderia manipular de maneira adequada.

A UML, por se tratar de uma linguagem, provê vocabulário e regras para representar seus elementos que descrevem um sistema. Esta linguagem define uma gramática que auxilia os desenvolvedores a criar e ler seus documentos. (Sommerville, 2005).

De acordo com Chonoles e Schardt (2003, p.12) uma analogia próxima pode ser a de plantas de engenharia, com detalhes suficientes para descrever como um prédio deve ser.

Seus modelos podem ser usados para gerar código escrito em diferentes linguagens de programação, tais como Java, C# e C++, além de tabelas de bancos de dados. Ferramentas Case, como o *System Architect*, *Jude* e *Together*, além de outros provêem este recurso.

A documentação gerada pela UML aborda tanto a arquitetura como todos os detalhes do sistema.

3.1. Características

As partes que compõem a UML são (Sommerville, 2005):

- **Caso de uso**: apresenta uma visão do ponto de vista externo do sistema, descrevendo seu comportamento sob a perspectiva dos usuários finais e projetistas do sistema;
- **Lógica**: descreve a arquitetura do sistema, ou seja, suas classes, interfaces, componentes e colaborações e identifica os tipos de serviços fornecidos pelo sistema;
- **Processo**: mostra o processamento concorrente do sistema através de processos e mecanismos de concorrência e sincronização;
- **Implementação**: correspondem aos componentes, subsistemas e arquivos que serão usados na composição do sistema;
- **Implantação**: o objetivo desta visão é fornecer informações sobre a distribuição e instalação das partes físicas do sistema.

3.2. Utilização

Segundo Sommerville (2005), a UML tem um vasto conjunto de utilização, sua principal função é a modelagem de regras de negócio e especificação de sistemas, abrangendo tanto os aspectos estruturais como os dinâmicos de um *software*.

É utilizada para modelar os mais diversos tipos de sistemas, fornecendo também diferentes visões. Pode ser aplicada em todas as fases do desenvolvimento, desde a análise de requisitos, até os testes finais, passando pela implementação e especificação técnica. Jacobson, Rumbaugh e Booch (2005).

Segundo Jacobson, Rumbaugh e Booch (2005, p.17), a UML se destina principalmente a sistemas complexos de *software*, tendo emprego de maneira efetiva em alguns casos, tais como: sistemas de informações corporativos, de telecomunicações, científicos e de serviços bancários e financeiros.

3.3. Principais Diagramas

Jacobson, Rumbaugh e Booch (2005) definem como sendo um diagrama a apresentação gráfica de um conjunto de elementos, geralmente representada como um gráfico conectado de vértices (itens) e arcos (relacionamentos).

Através da definição acima pode-se dizer que um sistema representa algo que está sendo elaborado, podendo ser visualizados de diversas formas e perspectivas e, para fazer essa representação, são utilizados os diagramas.

Segundo Jacobson, Rumbaugh e Booch (2005, p.95), para visualizar as partes estáticas de um sistema, utilizam-se os seguintes diagramas

- Diagrama de classes;
- Diagrama de componentes;
- Diagrama de estrutura composta;
- Diagrama de objetos;
- Diagrama de implantação;
- Diagrama de artefatos.

E, para visualizar as partes dinâmicas:

- Diagrama de caso de uso;
- Diagrama de seqüência;
- Diagrama de comunicação;
- Diagrama de gráfico de estados;
- Diagrama de atividades.

Os diagramas que mostram as partes estáticas de um sistema são nomeados como diagramas estruturais. E, os diagramas que mostram as partes dinâmicas de um sistema são nomeados diagramas comportamentais.

3.3.1. Diagramas de Classes

Jacobson, Rumbaugh e Booch (2005) definem o diagrama de classes como sendo um dos diagramas mais utilizados na modelagem de sistemas orientados a objetos por mostrar um conjunto de classes, interfaces e colaborações e seus relacionamentos. Os diagramas de classes também são utilizados como base para a construção de dois outros diagramas: os de componentes e os de implantação.

Como pode-se ver na figura 3.1, o diagrama de classes possibilita a visualização dos aspectos estáticos do sistema, mais especificamente de blocos de construção e seus relacionamentos e a especificação dos detalhes de construção. Jacobson, Rumbaugh e Booch (2005).

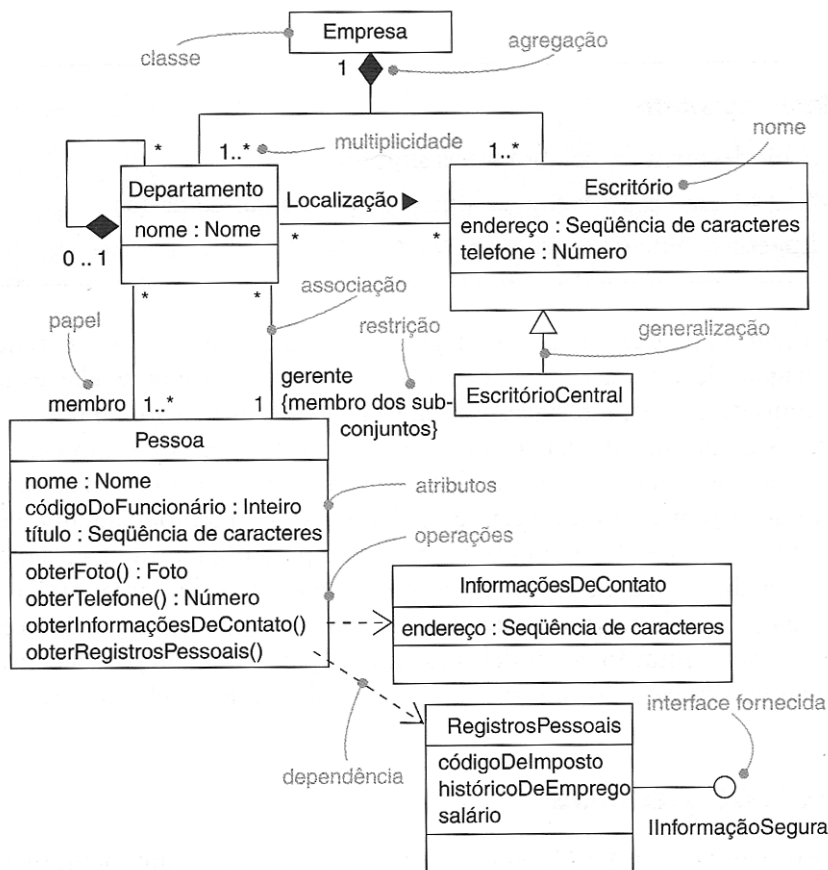



Figura 3.1 – Exemplo de Diagrama de Classes

Fonte: Jacobson, Rumbaugh e Booch (2005, p.108)

Segundo Jacobson, Rumbaugh e Booch (2005, p.109), a utilização do diagrama de classes está classificada em três formas:

- Para fazer a modelagem do vocabulário de um sistema;
- Para fazer a modelagem de colaborações simples;
- Para fazer a modelagem do esquema lógico de um banco de dados.

3.3.2. Diagrama de componentes

Jacobson, Rumbaugh e Booch (2005, p.97) definem um diagrama de componentes o diagrama que mostra as partes internas, os conectores e as portas que implementam um componente.

Um componente é um programa ou objeto reutilizável que executa dentro do sistema uma função específica ou pode ser criado para trabalhar com outros componentes já existentes.

3.3.3. Diagrama de estrutura composta

Jacobson, Rumbaugh e Booch (2005, p.97) definem um diagrama de estrutura composta o diagrama que apresenta a estrutura interna de uma classe ou colaboração. Esse diagrama possui pouca diferença em relação ao diagrama de componentes e Jacobson, Rumbaugh e Booch (2005) tratam os dois como diagrama de componentes.

3.3.4. Diagrama de objetos

O diagrama de objetos é definido por Jacobson, Rumbaugh e Booch (2005, p.188) como sendo o diagrama que faz a modelagem de instâncias de itens contidos em diagramas de classes. Em outras palavras, pode-se dizer que o diagrama de objetos mostra o conjunto de objetos e relacionamentos em determinado ponto no tempo.

Assim como o diagrama de classes, possibilita a visualização de maneira estática, porém, através de instâncias reais ou prototípicas (figura 3.2).

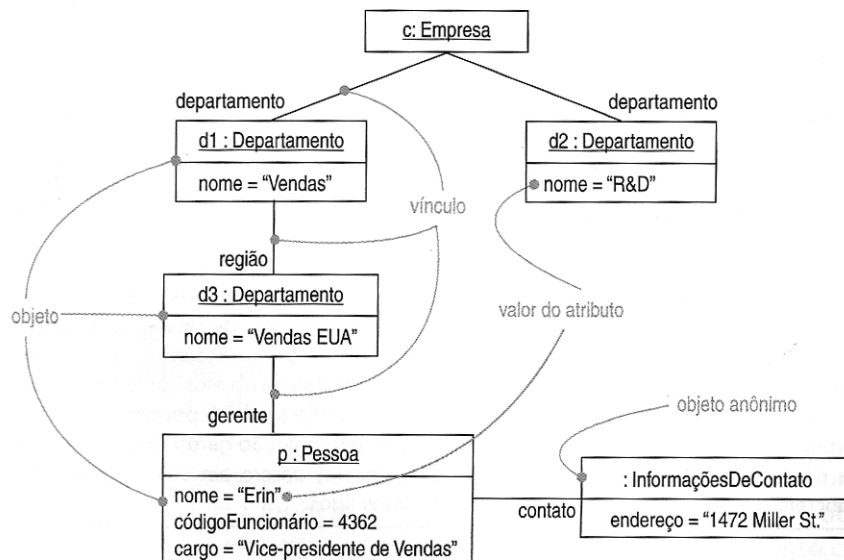



Figura 3.2 – Exemplo de Diagrama de Objetos

Fonte: Jacobson, Rumbaugh e Booch (2005, p.189)

3.3.5. Diagrama de implantação

Jacobson, Rumbaugh e Booch (2005, p.411) definem o diagrama de implantação como diagrama de classe que focaliza os nós do sistema, em que na maior parte envolve a modelagem da topologia de hardware em que o sistema é executado.

Em sua utilização, o diagrama de implantação é importante para a visualização, documentação e especificação de sistemas embutidos, cliente/servidor, distribuídos e até mesmo para o gerenciamento de sistemas executáveis por meio de engenharia reversa. (Jacobson, Rumbaugh e Booch, 2005). Pode-se ver um exemplo de diagrama de implantação na figura 3.3.

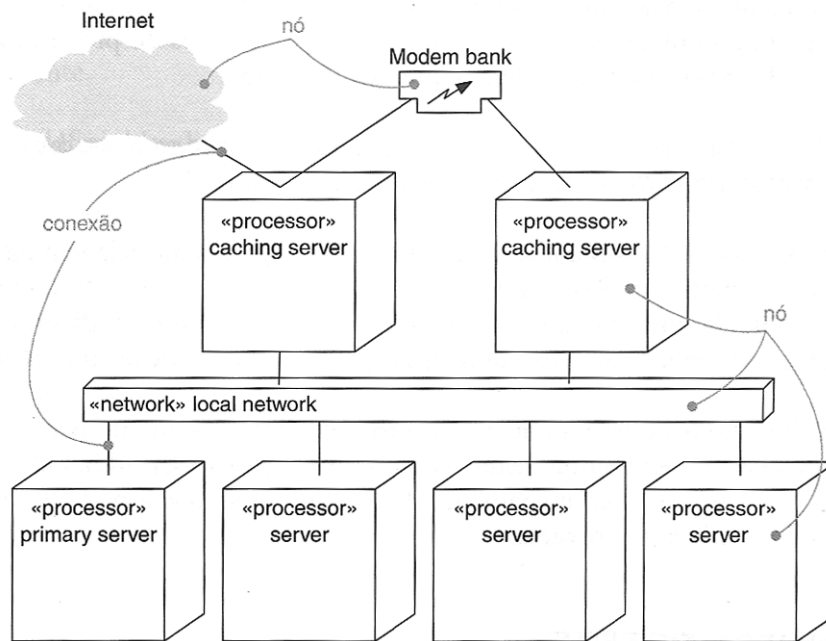



Figura 3.3 – Exemplo de Diagrama de Implantação

Fonte: Jacobson, Rumbaugh e Booch (2005, p.412)

3.3.6. Diagrama de artefatos

Jacobson, Rumbaugh e Booch (2005, p.397) define o diagrama de artefatos como sendo diagrama que mostra a visão estática de um sistema, envolvendo a modelagem de itens físicos residentes em um nó. Esses itens podem ser: programas executáveis, bibliotecas, tabelas, arquivos e documentos. É o diagrama de classes que focaliza os artefatos do sistema.

Segundo Jacobson, Rumbaugh e Booch (2005), o diagrama de artefatos é usado comumente para:

- Fazer a modelagem de código-fonte (figura 3.4);
- Fazer a modelagem de versões executáveis (figura 3.5);
- Fazer a modelagem de banco de dados físicos (figura 3.6);
- Fazer a modelagem de sistemas adaptáveis (figura 3.7).

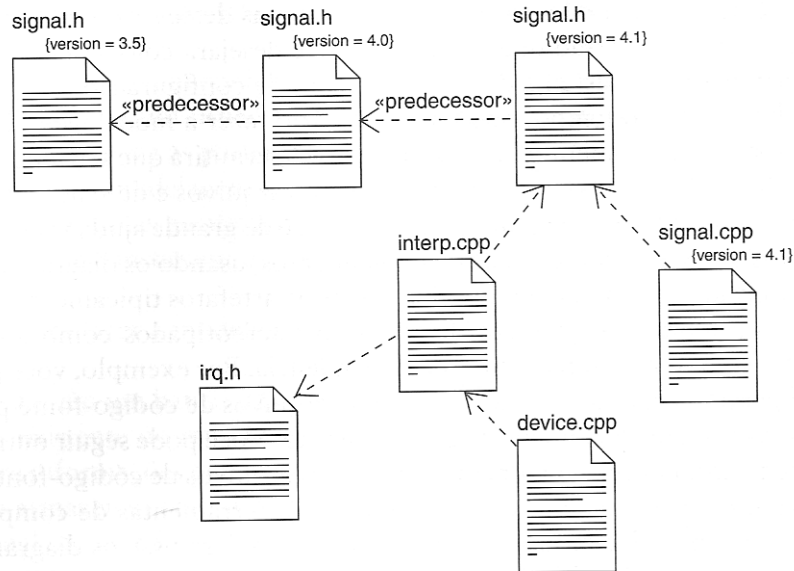



Figura 3.4 – Exemplo de Diagrama de Artefatos, modelagem de código-fonte

Fonte: Jacobson, Rumbaugh e Booch (2005, p.402)

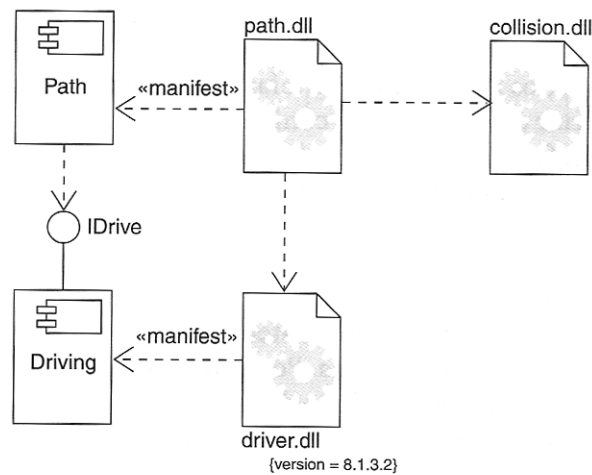



Figura 3.5 – Exemplo de Diagrama de Artefatos, modelagem de versões executáveis

Fonte: Jacobson, Rumbaugh e Booch (2005, p.404)

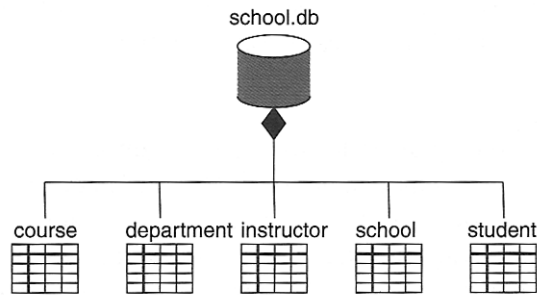



Figura 3.6 – Exemplo de Diagrama de Artefatos, modelagem de banco de dados físicos

Fonte: Jacobson, Rumbaugh e Booch (2005, p.406)

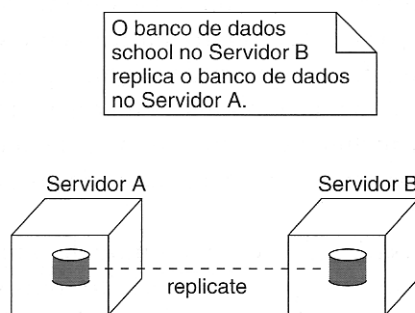



Figura 3.7 – Exemplo de Diagrama de Artefatos, modelagem de sistemas adaptáveis

Fonte: Jacobson, Rumbaugh e Booch (2005, p.407)

3.3.7. Diagrama de caso de uso

Segundo Jacobson, Rumbaugh e Booch (2005), nenhum sistema existe isoladamente. Todo sistema interage com atores humanos ou autômatos que usam o sistema com algum objetivo. Um diagrama de caso de uso identifica quais são os comportamentos pretendidos no desenvolvimento de um sistema.

Os casos de uso são um dos diagramas da UML que possibilitam a modelagem dos aspectos dinâmicos de um sistema. Através deles, os desenvolvedores podem compreender facilmente como o usuário final interage diretamente com o sistema.

Em geral, os diagramas de casos de uso costumam conter: Assunto, Casos de Uso, Atores e Relacionamentos, que podem ser de dependência, generalização e associação. Comumente, esse tipo de diagrama é utilizado para a modelagem do contexto de um assunto e para a modelagem dos requisitos de um sistema.

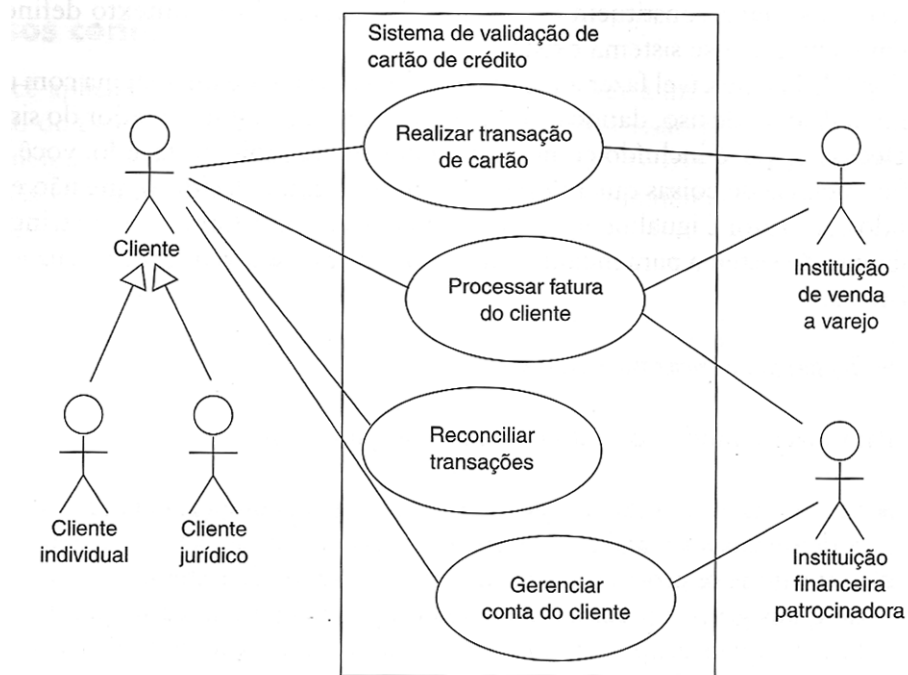



Figura 3.8 – Exemplo de Diagrama de Casos de Uso

Fonte: Jacobson, Rumbaugh e Booch (2005, p.246)

Jacobson, Rumbaugh e Booch (2005, p.250) sugerem que para um diagrama de caso de uso bem-estruturado é um diagrama que:

- Tem como foco comunicar um aspecto da visão estática de caso de uso do sistema;
- Contém somente os casos de uso e atores essenciais à compreensão desse aspecto;
- Fornece detalhes consistentes com seu nível de abstração;
- Não é tão minimalista, que informe mal o leitor sobre a semântica que é importante.

Para definir um diagrama de caso de uso, Jacobson, Rumbaugh e Booch (2005, p.250) também sugerem que:

- Dar um nome capaz de comunicar seu propósito;
- Distribuir seus elementos para minimizar o cruzamento de linhas;
- Organizar seus elementos espacialmente, de maneira que os comportamentos e papéis semanticamente relacionados apareçam próximos fisicamente;
- Usar notas e cores como indicações visuais e chamar atenção para características importantes do diagrama;

- Tente não mostrar muitos tipos de relacionamentos. Em geral, se existir relacionamentos de inclusão e estendido complicados, colocar esses elementos em outro diagrama.

3.3.8. Diagrama de seqüência

Segundo Jacobson, Rumbaugh e Booch (2005) um diagrama de seqüência enfatiza a ordem temporal das mensagens. Isso pode ser visualizado melhor na figura 3.9.

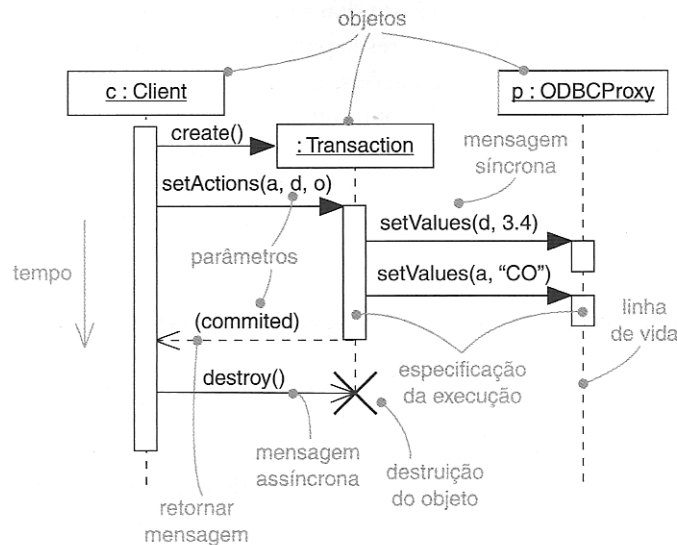



Figura 3.9 – Exemplo de Diagrama de Seqüência

Fonte: Jacobson, Rumbaugh e Booch (2005, p.255)

Existem duas características que diferenciam o diagrama de seqüência do diagrama de comunicação: a existência da linha de vida do objeto e do foco de controle. Observando a figura 3.9 pode notar uma linha tracejada na vertical, essa linha é a de vida do objeto, sendo que essa linha de vida deve ser visualizada de cima para baixo, da esquerda para a direita. Já o foco de controle é o retângulo que fica sobre a linha de vida do objeto. A parte superior do retângulo indica o início da ação e a parte inferior o fim.

As setas que vão da esquerda para a direita são mensagens que passam por cada foco de controle. As setas que vão da direita para a esquerda são as respostas que são passadas de um foco de controle para outro, assim como as mensagens.

3.3.9. Diagrama de comunicação

Jacobson, Rumbaugh e Booch (2005, p.259) definem diagrama de comunicação como o diagrama que enfatiza à organização dos objetos que participam de uma interação.

O diagrama de comunicação possui duas características que o diferem do diagrama de seqüência: a existência do caminho, na qual é representado o caminho correspondente a uma associação e do número de seqüência que indica a ordem temporal da mensagem. Pode-se observar essas características na figura 3.10.

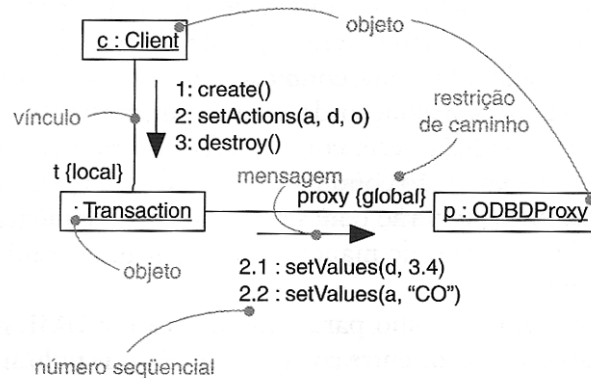



Figura 3.10 – Exemplo de Diagrama de Comunicação

Fonte: Jacobson, Rumbaugh e Booch (2005, p.259)

3.3.10. Diagrama de gráfico de estados

O diagrama de gráfico de estados é um dos cinco diagramas da UML que são utilizados para a modelagem da visão dinâmica de sistemas. Segundo Jacobson, Rumbaugh e Booch (2005), o diagrama de gráfico de estados permite a visualização do comportamento de objetos que é caracterizado por sua resposta a eventos ativados externamente do seu contexto, chamado de objetos reativos.

Como pode-se visualizar na figura 3.11, o diagrama de gráfico de estado enfatiza o fluxo de controle de um estado para o outro.

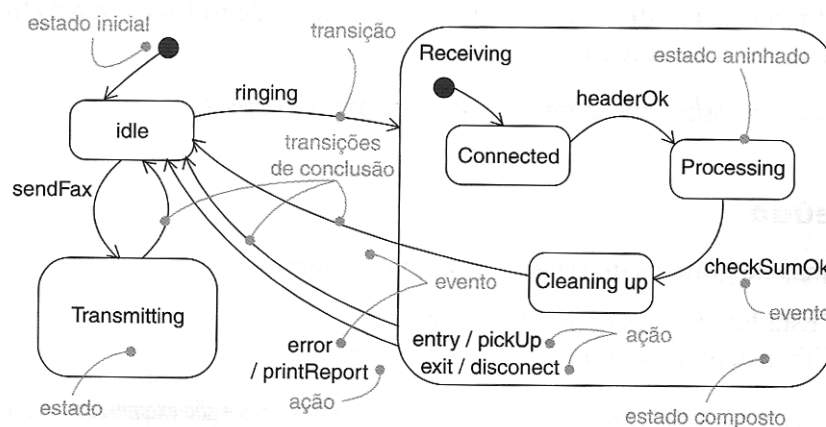



Figura 3.11 – Exemplo de Diagrama de Gráfico de Estados

Fonte: Jacobson, Rumbaugh e Booch (2005, p.343)

3.3.11. Diagrama de atividades

Assim como o diagrama de gráfico de estados, o diagrama de atividades é um dos cinco diagramas da UML que permite a visualização dos aspectos dinâmicos de um sistema. Diferentemente de um diagrama de gráfico de estados, o diagrama de atividades mostra também a concorrência de atividades existentes e as ramificações de controle. Tais características podem ser visualizadas na figura 3.12.

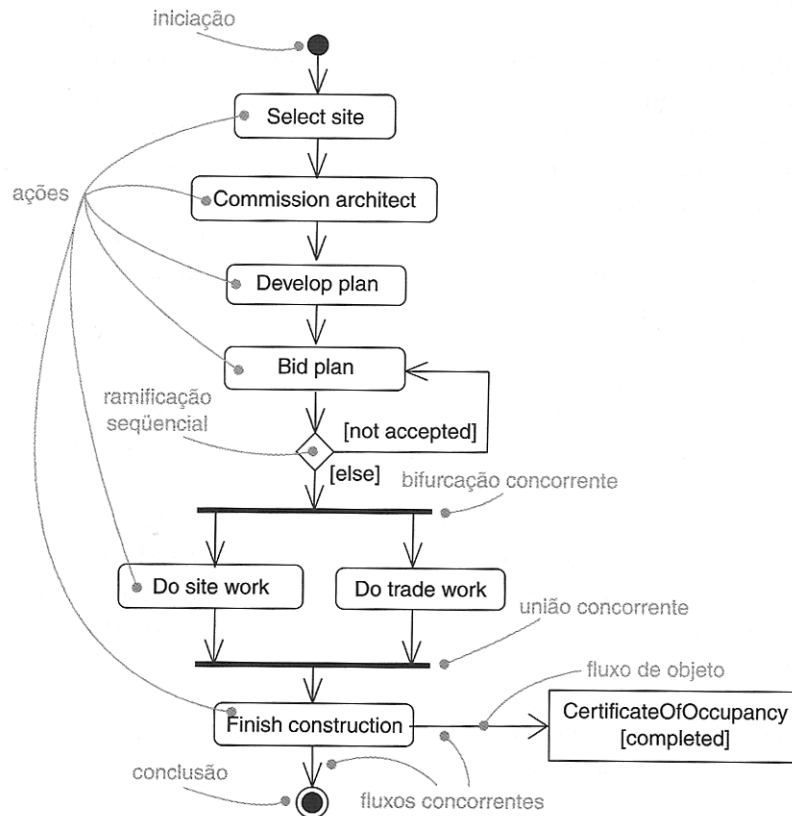



Figura 3.12 – Exemplo de Diagrama de Atividades

Fonte: Jacobson, Rumbaugh e Booch (2005, p.270)

Jacobson, Rumbaugh e Booch (2005, p.270) definem que um diagrama de atividades é essencialmente um fluxograma que enfatiza a atividade que ocorre ao longo do tempo.

CAP. IV – RUP - Rational Unified Process

Segundo Kruchten (2004, p.15) o *Rational Unified Processo*, RUP, é um processo de engenharia de software que fornece um modo disciplinado para assumir tarefas e responsabilidades dentro de uma empresa de desenvolvimento. Este processo é desenvolvido e mantido pela *Rational Software*, e esta disponível através da empresa IBM.

Este processo tenta resolver um grande problema, a dificuldade em fazer com que projetos de sistemas complexos sejam entregues nos prazos determinados. Conforme os anos vão passando, os sistemas vão ficando cada vez mais complexos, e precisando ser entregues em prazos mais curtos. Combinado a isso há novas tecnologias, que surgem a cada dia e a especificação do sistema se altera no decorrer do desenvolvimento. A utilização de uma metodologia tenta evitar que mudanças atrapalhem o andamento e conclusão do projeto no tempo determinado.

Segundo a Rational (2001), o RUP se divide em quatro fases diferentes que serão explicadas posteriormente:

- Concepção;
- Elaboração;
- Construção;
- Transição.

O RUP é composto de uma ou mais iterações, geralmente de período curto (uma a duas semanas), o que reduz o impacto de mudanças.

Além das fases e iterações, existem também os *workflows* que Kroll e Kruchten (2003, p. 38) nada mais são do que uma seqüência de tarefas relacionadas a um importante aspecto do projeto em questão. Cada tarefa é descrita em detalhes, onde é definido o responsável, a qual o *workflow* pertence e quais as entradas e saídas, que são descrições essenciais.

Segundo Kruchten (2004, p.5) o RUP segue as melhores práticas de desenvolvimento de *software*, entre elas:

- Desenvolvimento iterativo;
- Gerenciamento de requisitos;
- Arquitetura e uso baseados em componentes;
- Organização da especificação em modelos e UML;
- Verificação constante da qualidade;
- Controle de mudanças e configurações;

4.1. Desenvolvimento Iterativo

Segundo Kruchten (2004, p.6) o desenvolvimento iterativo proposto pelo RUP tem por objetivo conduzir o projeto em ciclos, isto é, iterações. Cada um destes ciclos é tratado de forma tradicio-

nal, ou seja, cada um tem a análise de requisitos e riscos, implementação, testes e implantação. Depois há outra iteração, na qual novos requisitos são tratados, outros riscos são considerados, há mais análise, testes e implantação, até que o projeto seja concluído ou cancelado.

Segundo Kruchten (2004, p.15) o problema básico do desenvolvimento de software com as abordagens clássicas é que o risco é adiado, tornando caro desfazer erros anteriores.

Esta abordagem permite um gerenciamento mais eficaz dos requisitos, levando-se em conta as constantes descobertas de necessidades que ocorrem ao longo do desenvolvimento do projeto. É mais fácil incluir novas idéias e necessidades ao projeto que é desenvolvido de forma gradual, produzindo-se novos artefatos a cada iteração de desenvolvimento. A equipe usa processos repetitivos e previsíveis que acaba convergindo para o objetivo final do projeto.

Kroll e Kruchten (2003, p. 32) definem que com esta abordagem há dois grandes benefícios:

- Permite que a equipe identifique os componentes que compõem o sistema de forma progressiva, possibilitando que decida-se quais serão desenvolvidos internamente, quais serão reaproveitados e quais serão comprados;
- A integração, ao final do projeto, não é considerada uma fase problemática, uma vez que os resultados de cada iteração são integrados progressivamente ao longo de seu desenvolvimento. O sistema final é composto por vários ciclos que ocorrem durante o projeto, eliminando assim grande parte dos riscos, que geralmente estão associados à fase de integração, permitindo, desta forma, aumentar significativamente as chances de sucesso.

Segundo Kroll e Kruchten (2003, p. 38) um artefato é uma parte de informação que é produzida, modificada ou usada por um processo. Artefatos são elementos tangíveis de projetos, que são usados como entrada de regras para executar uma atividade e são, também, o resultado de outras atividades.

4.2. Organização dos Modelos

Um modelo é criado tendo como base observações do mundo real, ou uma aproximação baseada nos objetivos do sistema.

O modelo deve ser simples o bastante para permitir uma rápida implementação da solução dos problemas levantados.

De acordo com Kroll e Kruchten (2003, p. 34) o RUP trabalha com os modelos providos pela UML:

- **Modelo de análise**: tem foco no negócio, organizando os elementos do Modelo em pacotes, que darão origem aos subsistemas;
- **Modelo de design**: refinamento do Modelo de análise, com foco na implementação física do sistema;
- **Modelo de banco de dados**: consiste nos modelos lógico e físico do banco de dados;

- **Modelo de casos de uso**: casos de uso organizados em pacotes, fornecendo uma visão externa do sistema;
- **Modelo de implantação**: descreve a distribuição física do sistema, indicando como os subsistemas serão distribuídos entre os nós da rede;
- **Modelo de implementação**: descreve como os elementos do sistema serão desenvolvidos;
- **Modelo de Negócio**: mostra como o negócio é visto pelos atores externos da UML;
- **Modelo de teste**: descreve como os componentes do Modelo de implementação serão testados.

4.3. Fases

No RUP, o projeto é composto por quatro fases: Concepção, Elaboração, Construção e Transição. Estas fases não têm que seguir uma seqüência tradicional que consiste em requisitos, análise, programação, integração e testes.

Segundo Kroll e Kruchten (2003, p. 35) cada uma das fases do RUP tem um marco e um conjunto de objetivos bem definidos. Deve-se fazer uso destes objetivos como um guia para a decisão de quais atividades devem ser levadas a diante e quais artefatos devem ser produzidos.

Cada fase termina em um marco relevante para o projeto como um todo, em que uma decisão importante deverá ser tomada, continuar e aprovar os recursos para a próxima fase ou cancelar o projeto.

Estas fases são executadas em um ou mais ciclos de desenvolvimento, abordagem esta que foi criada para ser dinâmica e adaptativa, podendo acomodar mudanças de objetivos e estratégias de desenvolvimento, muito comuns nos projetos de sistemas.

- **Concepção**: nesta fase a preocupação é estabelecer uma visão do sistema, do ponto de vista do negócio, permitindo avaliar se o projeto é ou não viável;
- **Elaboração**: o objetivo desta fase é capturar todos os requisitos ainda não identificados e definir uma arquitetura sólida que permita a evolução do sistema nas fases seguintes;
- **Construção**: nesta fase tanto o sistema quanto sua documentação são criados em conjunto a uma versão de testes do sistema, a primeira versão do manual do usuário e da ajuda. O manual de operação e instalação do sistema e o material para o curso de treinamento devem também começar a ser preparados nesta fase;
- **Transição**: garante que o *software* atenda às necessidades do usuário. Isto inclui testes do produto antes de seu lançamento, por meio de versões *beta* e ajustes baseados nas respostas do usuário, quanto à versão beta testada. Neste ponto do projeto o foco é a sintonia fina do produto, configuração, instalação e questões de usabilidade, qualquer grande questão estrutural já deve ter sido tratada bem antes desta fase.

4.4. Fluxos de Atividades

Os processos do RUP definem papéis, atividades, artefatos e os procedimentos que devem ser executados.

Os papéis estão associados a quem são as pessoas que irão executar o projeto, sendo que uma mesma pessoa pode executar mais de um papel no mesmo projeto. Estes papéis definem as responsabilidades e as tarefas que serão executadas e cada um requer um conjunto de competências por parte do responsável.

Como pode-se ver na figura 4.1, o eixo horizontal representa o tempo e mostra os aspectos do ciclo de vida do processo à medida que se desenvolve e o vertical representa as disciplinas, que agrupam as atividades de maneira lógica, por natureza.

As atividades representam pacotes de trabalho que produzem resultados relevantes para o projeto, geralmente associadas à criação ou atualização de artefatos, que por sua vez são os elementos a serem entregues ao cliente, tais como classes, componentes, subsistemas, interfaces, documentos, programas, executáveis, etc.

Segundo a Rational Software Corporation (2001) a seqüência de atividades resulta em artefatos a serem criados pelos responsáveis, de modo que, agreguem valor para o projeto. Um artefato pode ser traduzido como um diagrama de atividades UML, ou parte dele, e deve ser adaptado às várias fases do projeto, em detrimento dos seus objetivos. O RUP define nove fluxos de atividades (Figura 4.1.):

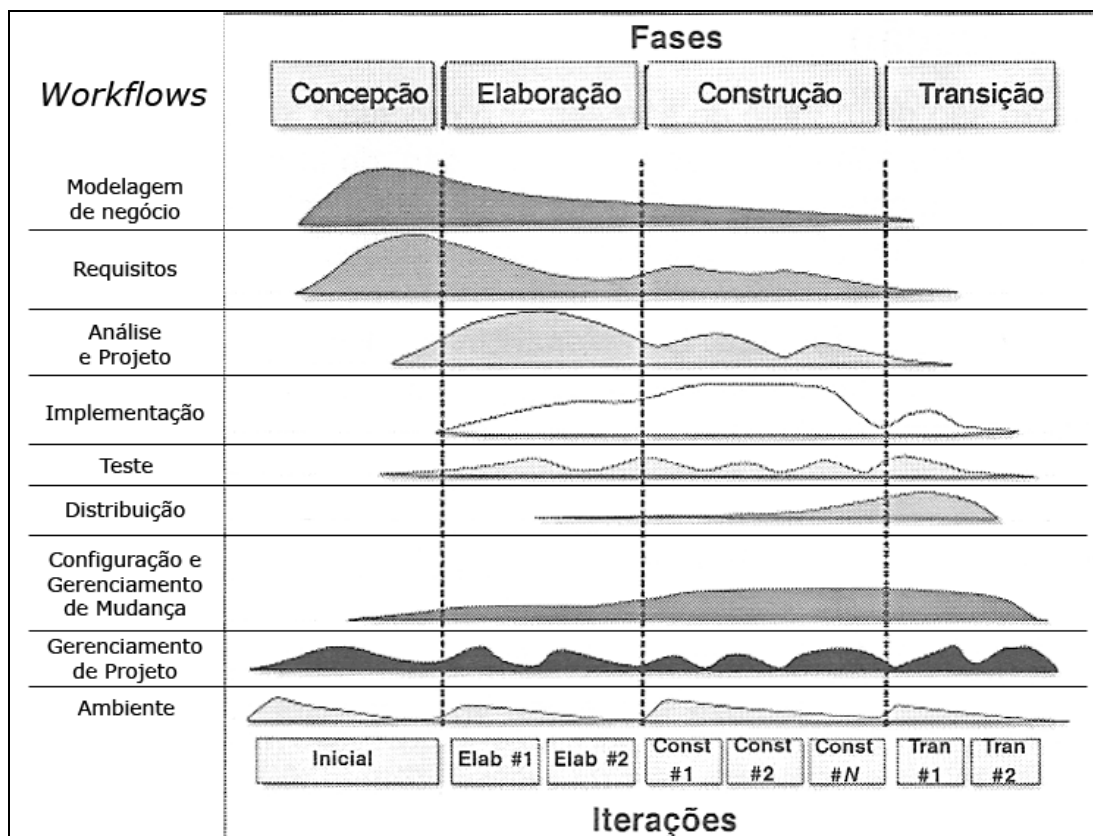



Figura 4.1. – Nove fluxos centrados no processo.
fonte: Philippe Kruchten (2004, p.36)

- Modelagem de negócio;
- Requisitos;
- Análise e design;
- Implementação;
- Testes;
- Distribuição;
- Gerenciamento de configuração e mudança;
- Gerenciamento de projeto;
- Ambiente

4.4.1. Modelagem do Negócio

Onde são documentados os conceitos relativos ao negócio. Este documento será refinado posteriormente, à medida que os casos de uso forem sendo detalhados.

Segundo Kruchten (2004, p. 122) este modelo consiste em atores, que representam papéis externos para o negócio e os casos de uso são os negócios.

Os processos são documentados através de casos de uso, enquanto as entidades pertinentes são representadas através de diagramas de Entidade Relacionamento. Este modelo mostra quem são os participantes do negócio, os *workflows*, as atividades, as pessoas e suas funções.

Segundo a Rational (2001) o modelo de negócio tem os seguintes objetivos:

- Modelar o negócio como ele é atualmente;
- Descobrir redundâncias de atividades nos processos;
- Identificar os gargalos;
- Localizar possibilidades de melhorias nos processos.

Os artefatos que são trabalhados neste processo são os seguintes:

- Documento da Visão do Negócio;
- Modelo de Casos de Uso de Negócio;
- Modelo de Objetos do Negócio;
- Regras do Negócio;
- Glossário do Negócio.

4.4.2. Requisitos

O processo de detecção de requisitos tem vários objetivos, tais como estabelecer as funções que o sistema deve desempenhar, esboçar a interface de usuário, fornecer uma tradução clara dos requisitos à equipe de desenvolvimento e também do sistema, assim como prover informações para o planejamento do projeto, tais como conteúdo técnico para os ciclos de desenvolvimento, além de subsídios para estimativas de prazo e custo.

Segundo Pressman (1995, p. 266) a função e o desempenho atribuídos ao software são refinados quando se estabelece uma descrição completa da informação, indicação dos requisitos de desempenho e restrições de projeto [...].

Os requisitos podem ser funcionais, aqueles capturados através dos casos de uso, que documentam as entradas, as saídas e os processos; assim como não funcionais, que são compostos por características não associadas ao comportamento esperado do sistema, como usabilidade, confiabilidade, desempenho e suporte.

Neste processo, as pessoas envolvidas vão desempenhar os seguintes papéis:

- Analista de sistemas: o profissional que comanda o processo;
- Especificador de Casos de Uso: o profissional que detalha as funcionalidades do sistema através dos casos de uso;
- Projetista de Interface de Usuário: o profissional que vai desenvolver os protótipos das interfaces com o usuário.

Os artefatos desenvolvidos e atualizados neste processo são os que definem os requisitos e o escopo do sistema.

Artefatos trabalhados pelo Analista de Sistemas:

- Plano de gerenciamento de requisitos;
- Glossário de termos do negócio e do sistema;
- Visão do software;
- Especificações suplementares;
- Modelo de Casos de Uso.

Artefatos trabalhados pelo Especificador de Casos de Uso:

- Casos de Uso;
- Pacote de Casos de Uso;
- Especificações dos requisitos do *software*.

Artefatos trabalhados pelo Projetista de Interface com o Usuário:

- Ator;
- Classes de fronteira;
- Roteiro dos Casos de Uso;
- Protótipo da Interface com o Usuário.

4.4.3. Análise e Design

O objetivo deste processo é traduzir os requisitos numa especificação que explica como o sistema deve ser desenvolvido, por exemplo, características como a arquitetura, o ambiente operacional e a escalabilidade. Este processo aborda tanto a análise quanto o design do sistema.

Segundo Kruchten (2004, p. 146) o modelo de análise omite a maioria dos detalhes do sistema e fornece uma avaliação de suas funcionalidades.

- **Análise**: esta especificação faz abstrações, evitando resolver problemas que serão resolvidos no processo de design e visualiza apenas as entidades associadas ao negócio. Esta separação permite uma melhor elaboração de especificações, já que não considera as limitações técnicas associadas à tecnologia;
- **Design**: esta especificação foca na produção de uma visão detalhada do sistema, considerando a tecnologia a ser usada no desenvolvimento, isto significa, a linguagem de programação, o sistema gerenciador de banco de dados, sistemas operacionais e outros detalhes técnicos.

O sistema é dividido em subsistemas, que se comunicam através de interfaces bem definidas. O modelo de classes composto inicialmente pelas classes de análise é expandido, incluindo as classes necessárias para a implementação do sistema como um todo. Os atributos de todas as classes são detalhados considerando-se a linguagem que será utilizada e os tipos de dados são identificados. O acesso aos métodos e propriedades são indicados, assim como a forma de encapsulamento dos subsistemas.

Neste processo os profissionais, segundo Kruchten (2004, p.145), vão desempenhar os seguintes papéis:

- Arquiteto: lidera e coordena as atividades técnicas do projeto;
- Projetista: define as responsabilidades, atributos e relações entre as classes;
- Projetista de Banco de Dados: especifica os modelos lógicos e físicos do banco de dados;
- Revisor da arquitetura e do design: revisa os artefatos produzidos durante o processo.

Os artefatos trabalhados pelo Arquiteto:

- Modelo de Análise;
- Modelo de Design;
- Interfaces;
- Descrição da arquitetura de *software*;
- Artefatos de tempo real.

Artefatos trabalhados pelo Projetista:

- Realização de Casos de Uso;
- Classes;
- Pacote de design;
- Especificações de subsistemas.

Artefato trabalhado pelo Projetista de Banco de Dados:

- Modelos de dados: lógico e físico.

4.4.4. Implementação

Durante este processo serão desenvolvidas várias versões do sistema, ou partes dele, de modo que representem sua funcionalidade. Cada uma destas versões é construída através da integração de componentes novos ou subsistemas. Os componentes são construídos separadamente e posteriormente combinados, criando subsistemas, que por sua vez são combinados, montando um sistema. A última versão, a operacional, é a que será instalada no cliente.

Segundo Kruchten (2004, p. 158) a meta é assegurar que este modelo seja organizado de tal maneira que faça o desenvolvimento de componentes e o processo de construção tão livres de conflitos quanto possível.

Este processo tem quatro objetivos principais:

- Definir a organização do código fonte no tocante ao desenvolvimento dos subsistemas;
- Implementar classes, objetos e componentes;
- Executar testes de unidade nos componentes desenvolvidos;
- Integrar os elementos implementados pelos programadores em arquivos executáveis.

Os seguintes artefatos são trabalhados neste processo:

- Subsistemas implementados;
- Componentes;
- Plano de integração das versões.

4.4.5. Teste

A fase de testes não é uma atividade isolada, nem uma fase do projeto na qual se executam todos os testes. Se os desenvolvedores desejarem ter retorno sobre a programação em tempo real, devem executar testes a cada ciclo de desenvolvimento do projeto.

Segundo Pressman (1995, p. 789) testes são realizados e todos os resultados são avaliados, ou seja, são comparados com os resultados esperados. Quando dados errôneos são encontrados infere-se a presença de erros e inicia-se a depuração.

Pode-se testar a funcionalidade dos protótipos, sua estabilidade e desempenho, ainda com a possibilidade de corrigir eventuais erros em tempo de desenvolvimento, isso além de assegurar a qualidade do produto entregue ao cliente.

A idéia de testar tudo apenas quando o sistema estiver pronto desconsidera o principal benefício do teste, detectar problemas enquanto ainda há tempo para se fazer algo a respeito.

Os seguintes artefatos são trabalhados neste processo:

- Plano de Testes;
- Modelo de Testes, composto por:
    - Casos de Teste;
    - Procedimentos de Teste;
    - Scripts de Teste;
    - Notas.
- Resultados;
- Modelo de Carga.

4.4.6. Distribuição

O objetivo deste processo é disponibilizar o sistema para os usuários. O maior nível de atividades deste processo ocorre na fase de transição, entre o novo sistema e o anterior, caso exista. Para facilitar os trabalhos o usuário deve ser envolvido o mais cedo possível, iniciando com a avaliação de versões beta e apontando procedimentos que pode vir a facilitar esta transição.

Distribuição consiste em:

- Teste do sistema no ambiente de produção;
- Empacotamento do software para distribuição;
- Distribuição do software;
- Instalação do software;
- Treinamento dos usuários e equipe comercial;
- Migração de dados para o novo sistema.

Os artefatos trabalhados neste processo são:

- Plano de Implantação;
- Software executável;
- Artefatos de Instalação;
- Documentação;
- Material de treinamento.

4.4.7. Gerenciamento de Configuração de Mudança

O objetivo deste processo é controlar as mudanças solicitadas para os artefatos do sistema, decorrentes de correções de erros, melhorias e inclusão de novos requisitos ou necessidades. Todas as solicitações devem ser analisadas, registradas, recusadas ou aprovadas e desenvolvidas.

Caso isto ocorra deverá ser feita uma análise de impacto e redefinição das prioridades.

Segundo Kruchten (2004, p. 175) os membros da equipe devem identificar e localizar artefatos, selecionar a versão apropriada, conhecer sua história para entender seu estado atual e as razões para ter mudado e identificar seu atual responsável.

Num processo iterativo os artefatos evoluem e são atualizados constantemente, os programadores, quando necessário, devem ter condições de localizar suas várias versões com facilidade.

Os seguintes artefatos são trabalhados neste processo, de acordo com a Rational (2001):

- Plano de Gerenciamento de Mudanças;
- Solicitações de Mudança;
- Modelo de Implementação.

4.4.8. Gerenciamento de Projeto

Segundo Kruchten (2004, p.95) gerenciamento de projeto de software consiste em balancear objetivos concorrentes, gerenciando o risco e todas as restrições do projeto para entregar um produto que atenda as necessidades do cliente. O fato de pouco projetos terem 100% de sucesso é um indicador de como é difícil realizar esta tarefa.

Este modelo tem três propósitos:

- Prover uma estrutura para gerenciar projetos de software;
- Prover guias para planejamento, gerenciar as pessoas envolvidas no projeto, executá-lo e monitorá-lo;
- Prover uma estrutura de gerenciamento de riscos.

No entanto esta disciplina do RUP não tem por objetivo cobrir todos os aspectos do gerenciamento de projetos, tais como contratação de pessoal, treinamento, gerenciamento de orçamento, contratos com fornecedores nem clientes. O objetivo desta disciplina consiste em planejar um projeto iterativo através do seu tempo de vida, gerenciamento de riscos e monitoração do progresso e as medições do progresso iterativo do projeto.

4.4.9. Ambiente

Este processo tem por objetivo definir os processos e ferramentas que proporcionem a implementação do sistema. Deve-se avaliar a cultura atual da empresa, definir-se uma lista de ferramentas que podem ser utilizadas e modelos de documentos que serão necessários.

Segundo Kruchten (2004, p. 190) a meta do fluxo de ambiente é fornecer suporte adequado para a organização do desenvolvimento em ferramentas, processos e métodos.

É neste processo que é definida a sistemática de seleção, aquisição, instalação e configuração de ferramentas pertinentes ao processo de desenvolvimento, criação e evolução dos processos, manutenção de infra-estrutura, procedimentos de cópias de segurança e outras rotinas pertinentes ao desenvolvimento.

Artefato trabalhado neste processo:

- Modelo de Desenvolvimento.

CAP. V – Projeto

Devido aos fatos anteriormente mencionados, neste trabalho será desenvolvido um Sistema de Gerenciamento de Projetos com características e funcionalidades semelhantes aos demais existentes, porém com foco diferenciado.

Como o objetivo é estudar a metodologia RUP com desenvolvimento orientado a objetos, não será desenvolvida uma ferramenta complexa de gerenciamento de projetos, e sim, será abordado de forma acadêmica as dificuldades que os *stakeholders* enfrentam em seu cotidiano.

Neste capítulo será explorado como o projeto foi desenvolvido.

5.1. Fase Concepção

O objetivo desta fase é a identificação do que será construído, quais serão as funcionalidades do sistema, determinar pelo menos uma possível solução, definir o prazo, o custo e os riscos, decidir qual processo seguir e quais ferramentas e tecnologias serão utilizadas.

Nesta fase foram elaborados os seguintes artefatos:

- Documento de Visão (ver Anexo A);
- Caso de Uso (ver Anexo A).

5.1.1. Documento de Visão

A elaboração deste documento deu-se por ser o marco inicial do projeto. Neste artefato foram descritos os *stakeholders* envolvidos, assim como, suas responsabilidades e a definição dos usuários que utilizarão o sistema. Além disso, foram definidos os requisitos funcionais e não funcionais tais como: segurança e disponibilidade. Este artefato define a visão que as partes envolvidas tem do sistema, suas necessidades e características importantes.

Este documento foi atualizado e mantido durante todo o tempo de vida do projeto e à medida que novos requisitos foram apontados, detalhados e inseridos no escopo do projeto, o Documento de Visão refletiu estas mudanças.

Um ponto importante neste artefato é a descrição da visão geral do sistema proposto. Neste item, são abordados os objetivos do sistema, que serve como base para próximo o artefato, o Casos de uso.

A finalidade deste artefato foi coletar, analisar e definir os requisitos do sistema. O foco está nos recursos necessários dos envolvidos no desenvolvimento, os usuários do sistema e as razões que levam a estas necessidades.

5.1.2 Casos de uso

Os casos de usos são documentos importantes no processo de desenvolvimento de um sistema. Este artefato é uma seqüência das ações realizadas por um sistema que produz um resultado de valor observável para determinado ator. Estes documentos são geralmente escritos pelos analistas do projeto, que definem as funcionalidades que o futuro sistema terá.

Os casos de usos foram levantados, tendo como base, os sistemas de gerenciamento de projetos já existentes. Foram identificadas as funcionalidades em comum e através disto, os ca-

sos de usos foram descritos. Também foram incorporados casos de usos que representam relatórios no sistema, conforme estudos realizados sobre gerenciamento de projetos.

Para cada caso de uso, escreveu-se uma breve descrição e um fluxo de informações, o que chamamos de fluxo básico. Além dos fluxos básicos, podem ocorrer os fluxos alternativos, caso o fluxo básico não seja seguido corretamente, por isso, os fluxos alternativos também foram identificados e escritos.

Também foram descritas as sessões de condições prévias e condições posteriores. As condições prévias são necessárias para que o sistema possa executar o caso de uso, enquanto que as condições posteriores são partes que descreve-se o que será executado após o término do caso de uso. Para representar graficamente este documento, utiliza-se os recursos da UML de uma ferramenta *Case* (Enterprise Architect v4.1) para criar-se os diagramas.

Este documento foi utilizado pela equipe de desenvolvimento para realizar os testes, verificando se as funcionalidades foram implementadas corretamente, para entender o comportamento e aprovar os fluxos de eventos do sistema.

5.2. Fase Elaboração

O objetivo desta fase é assegurar que a arquitetura, os requisitos e os planos sejam estáveis o suficiente. Nesta fase os riscos devem ser diminuídos a fim de determinar com segurança o custo e a programação para a conclusão do desenvolvimento e produzir um protótipo navegável do sistema.

Nesta fase foram desenvolvidos os seguintes artefatos:

- Protótipo navegável;
- Documento de arquitetura de software;
- Modelo de design;
- Modelo de dados;
- Guia de programação.

5.2.1. Protótipos

O protótipo tem como objetivo fornecer uma aparência visual do sistema. Os aspectos analisados na produção do protótipo navegável foram: definição de padrões de layout (como páginas de cadastro e de cadastro) e utilização de cores.

Inicialmente a prototipação do sistema foi elaborada no *software* Access 2003 da empresa Microsoft.. Esta decisão foi tomada, pelo fato do *software* possuir vários recursos para se montar um modelo de página rapidamente. Neste momento foram concentrados os esforços para identificar como ficaria a disposição dos componentes no formulário e, quais informações de texto seriam postas nas páginas (ver Figura 5.1). Depois de definido essas variáveis, o protótipo foi reconstruído para portar a plataforma web. Tecnologias como o HTML (*Hyper Text Markup Language*), CSS (*Cascading Style Sheets*) foram utilizadas para dar a visualização, navegação e usabilidade do sistema (ver Figura 5.2).

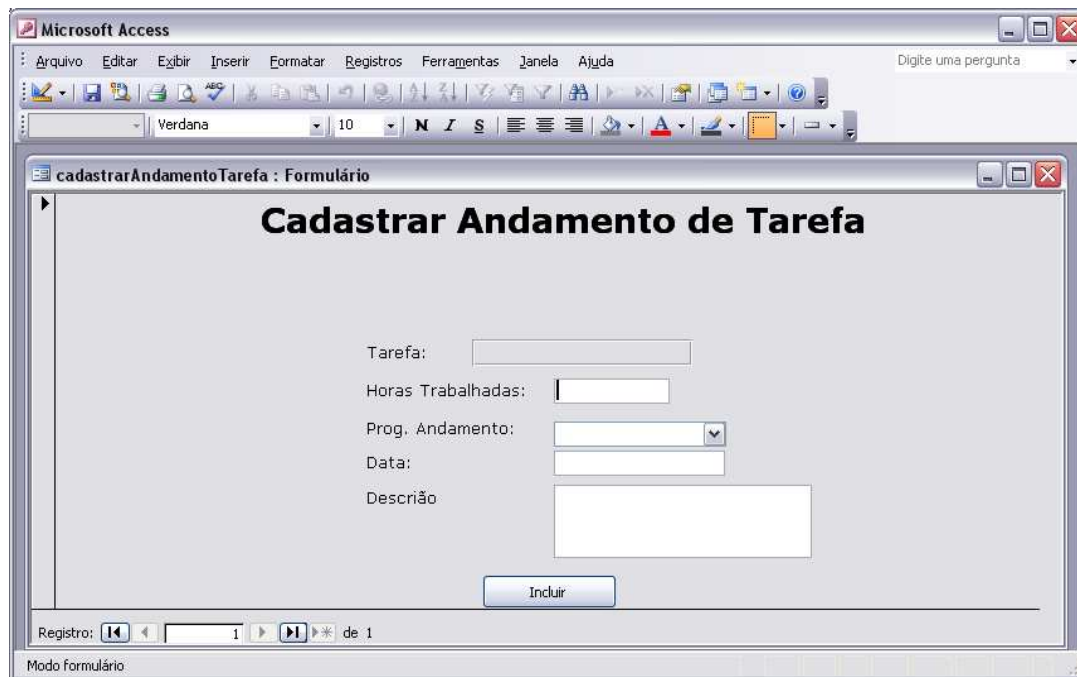



Figura 5.1. – Comparativo de sistemas de gerenciamento de projetos.

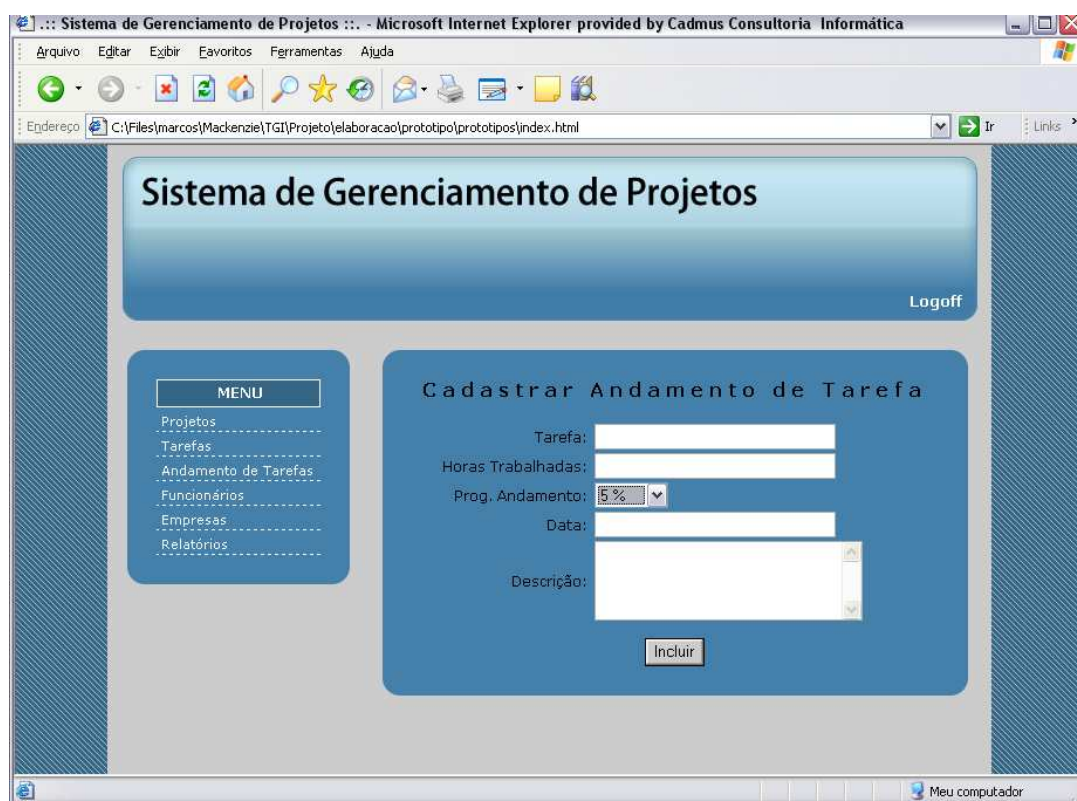



Figura 5.2. – Comparativo de sistemas de gerenciamento de projetos.

5.2.2. Arquitetura de Software

Este documento fornece uma visão geral da arquitetura que pode modelar os diferentes aspectos do sistema. É utilizado também, para a comunicação do arquiteto de software com os outros membros do projeto.

A criação da hierarquia de pacotes foi fundamental neste documento quando foi definida a visão lógica do sistema. O intuito era dividir e organizar as classes que seriam implementadas posteriormente e por isso, os pacotes foram criados seguindo um padrão conforme o exemplo abaixo:

net.sgp.empresa - Pacote responsável pelo cadastro de clientes (empresas).

Neste artefato foram descritos também a representação, objetivos e limitações da arquitetura. A representação da arquitetura esta baseada na arquitetura de referência JEE (*Java Enterprise Edition*) da empresa Sun Microsystems. Devido a este fato muitos dos mecanismos técnicos adotados, como persistência, ambiente e distribuição utilizarão esta arquitetura de referência.

O documento de arquitetura de software contemplou também as visões de implementação e implantação, onde procurou representar graficamente como o sistema é dividido em camadas (implantação) e como estão alocas as classes (implementação).

5.2.3. Modelo de Design

O modelo de design é uma abstração da implementação do sistema proposto. Nele foram definidas as realizações dos Casos de usos e as classes estáticas do sistema. As classes estáticas do sistema foram representadas graficamente no diagrama de classes (recurso da UML), conforme anexo B.

As entidades Empresa, Funcionário, Projeto, entre outras, e suas relações entre si de agregação e composição foram identificadas. Para cada entidade, foram identificados os seus atributos e seus tipos de dados (como Integer, Double e String). A criação deste diagrama foi baseada na leitura e interpretação do documento de caso de uso da fase de Concepção. Somente através dos Casos de usos pode-se levantar as entidades relevantes e, a partir daí, identificar os seus atributos. Nem todos os atributos estão de forma explícita. Alguns como o código do Projeto foram colocados no intuito de ter um atributo único no objeto instanciado, para diferenciar de outros objetos de mesmo tipo.

Outro item importante neste documento foram as realizações, também conhecidas como diagramas de seqüência. O diagrama de seqüência também é um recurso da UML e visa ter um panorama da implementação dos Casos de usos. Para cada Caso de uso implementado, o diagrama mais importante que os desenvolvedores tinham em mãos era o de seqüência. Neste diagrama estão previstos os objetos e suas trocas de mensagem (métodos) em tempo de execução, assim como as passagens de parâmetros necessárias para se ter algum processamento no sistema.

Nestas realizações, foi aplicado um *pattern* para que o sistema fosse inteiramente padronizado possibilitando que o código-fonte ficasse entendível e, que os objetos não tivessem muita ligação, ou seja, acoplamento. O *pattern* escolhido foi o MVC (*Model View Controller*), que define uma camada de apresentação (JSP – *Java Server Pages*, Servlet), outra de controle (EJB – *Enterprise Java Beans* e objetos de negócio) e a última de persistência (acesso ao banco de dados).

Cada camada tem um objetivo específico:

- A camada de apresentação é responsável em receber as solicitações dos usuários e direcioná-las aos objetos de negócios. Após o processamento das informações, os resultados são mostrados ao usuário na tela do *browser*. É nesta camada que o usuário interage com o sistema.
- A camada de controle é responsável em controlar toda a transação que foi ativada por um usuário no sistema. Esta camada recebe as solicitações da camada anterior e as direciona para a camada de persistência, que por sua vez, é responsável em atualizar as informações no banco de dados ou em algum sistema legado. Com base nesse *pattern*, o sistema foi implementado conforme o diagrama a seguir:

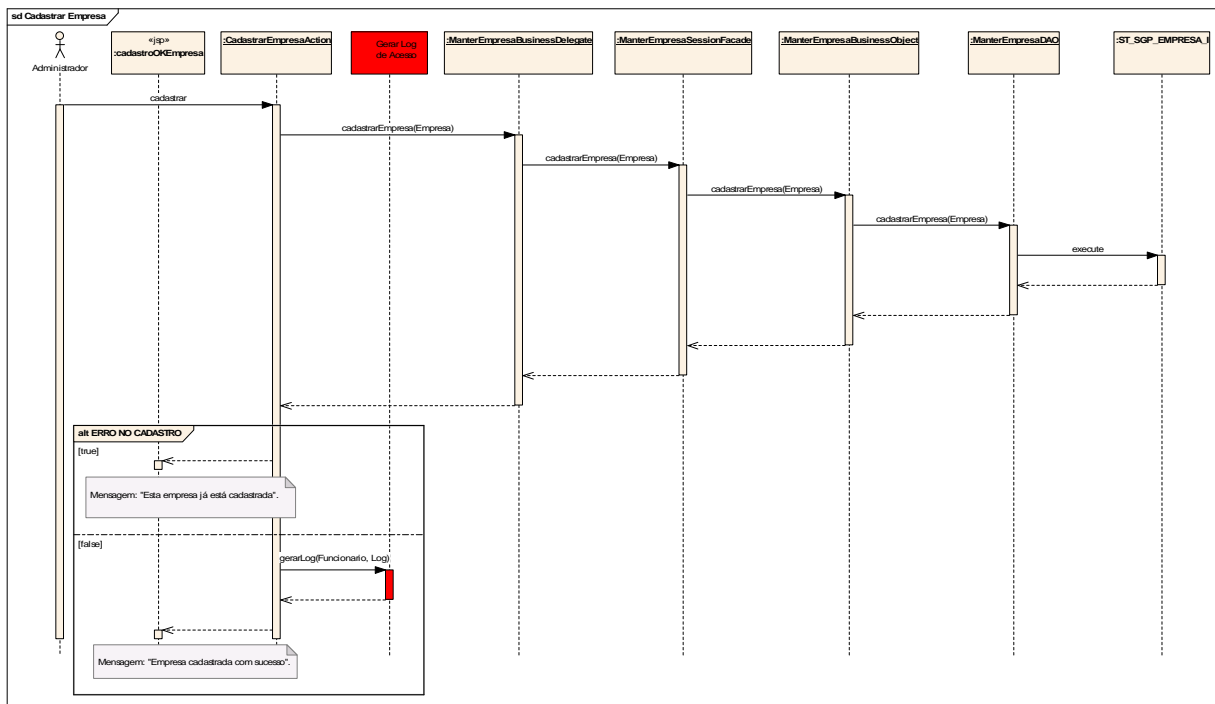



Figura 5.3. – Realizações.

A figura 5.3 apresenta os seguintes componentes:

- **ST_SGP_EMPRESA_I** é uma storage procedure (ST) que são códigos SQLs (Structured Query Language).
- **ManterEmpresaDAO** é responsável pelas conexões aos banco de dados e executar as storage procedures. Este objeto faz parte da camada de persistência.
- **ManterEmpresaBusinessObject** é o objeto responsável pela regra de negócio. Toda regra de negócio é codificada neste objeto.
- **ManterEmpresaSessionFacade** é o objeto responsável em controlar a transação.
- **ManterEmpresaBusinessDelegate** é o objeto responsável em localizar o objeto ManterEmpresaSessionFacade que pode estar no mesmo ou em outro servidor.

- **CadastrarEmpresaAction** é o objeto responsável em receber dos dados do browser.
- **cadastrarOKEmpresa.jsp** é a página que o cliente interage, seja para criar uma solicitação ou para receber alguma informação.

Todas as realizações foram feitas com base nesse padrão. O restante dos diagramas do sistema pode ser visualizado nos anexos B.

5.2.4. Modelo de Dados

O modelo de dados representa como as informações do sistema que serão armazenadas no banco de dados. Para isso, criou-se o MER (Modelo Entidade Relacionamento) a partir do diagrama de classe do artefato anterior, que identifica as tabelas, colunas e seus tipos de dados e os relacionamentos e suas associações.

Neste documento, o importante é criar um modelo consistente e organizado para que os dados sejam armazenados. Para não ter nenhuma dúvida dos campos inseridos nas respectivas tabelas, foram dicionarizadas todas as colunas, explicando-se o que estas representam nas tabelas e foram definidas as chaves PK (Primary Key) e FK (Foreign Key) das tabelas para que o sistema não tenha dados duplicados.

Para criar o modelo, utilizou-se uma ferramenta *Case* própria de banco de dados que auxiliou e muito na geração automática dos scripts SQLs para a criação das tabelas no sistema.

5.2.5. Guia de Programação Java

O guia de programação java é um manual de boas práticas de programação na linguagem. Este documento auxiliou os desenvolvedores para se criar um padrão de codificação para que qualquer desenvolvedor possa continuar um trabalho outrem sem problemas de adaptação.

Este guia é um manual de boas práticas baseados em padrões e experiências de desenvolvimento de profissionais da área. O documento auxilia qualquer desenvolvedor a padronizar nomes de atributos, métodos, métodos de acesso, passagem de parâmetros dos métodos, constantes, classes interfaces, variáveis locais. Auxilia também em como estruturar o código-fonte do sistema para o tratamento de exceções e de importação na utilização de pacotes próprios da linguagem ou dos pacotes criados no sistema.

É um manual de orientação básica que foi seguido a fundo na fase de construção com objetivo de ganhar produtividade no desenvolvimento do sistema.

5.3. Fase Construção

O objetivo desta fase é codificar os requisitos para que estes se tornem um programa. A fase de construção concentrou-se em gerar o sistema que até então estava sendo analisado e projetado.

Inicialmente esta fase constitui-se basicamente da montagem do ambiente e codificação dos Casos de usos. Para que nesta fase não fosse encontrados muitos problemas optou-se em criar um plano de desenvolvimento de software, no qual, a partir das fases anteriores, foi determinada a seqüência cronológica das atividades.

Neste plano de desenvolvimento, a construção do sistema seguiu a seguinte ordem:

1. Criação do banco de dados. A partir do MER (Modelo Entidade Relacional) foram gerados os *scripts* SQL (*Structured Query Language*).
2. Criação das *procedures*. Nesta etapa foram criadas todas as consultas de acesso ao banco de dados.
3. Carga na tabela PERMISSAO. Esta tabela contempla todas as permissões que o sistema de gerenciamento de projetos terá. As permissões estão descritas no caso de uso (UC02 – Vincular Permissões), conforme anexo A (Caso de Uso).
4. Montagem do ambiente de desenvolvimento. Nesta atividade foi criado na ferramenta de desenvolvimento (Eclipse) o ambiente do desenvolvedor. Este ambiente compreende-se basicamente na construção do arquivo (*build.xml*) responsável em compilar e gerar o programa executável, além de copiar e compactar os arquivos do sistema como imagens, javascripts, arquivos do java, arquivos XML (*Extended Markup Language*), entre outros. Neste ambiente, também foi criado a estrutura de pacotes, que está de acordo com o documento de arquitetura que se encontra no anexo B. Como o programa utilizou o banco de dados MySQL 5.0 e o *framework* de apresentação Struts da empresa Jakarta, os arquivos JAR (arquivos compactados) tiveram que ser incorporados no *classpath* do Eclipse e ao projeto. Com o ambiente praticamente terminado, as estruturas das classes foram criadas nos seus respectivos pacotes.
5. A partir daí, os Casos de usos foram codificados. Primeiramente foram implementados os Casos de usos UC07 – Realizar Login e UC08 – Verificar Permissões para que fosse possível utilizar o sistema. Depois, o Caso de uso UC09 – Gerar Log de Acesso foi incorporado ao sistema. O restante dos Casos de usos foram codificados posteriormente, pois estes possuíam dependência dos anteriormente citados. Alguns casos de usos como o UC01 – Manter Funcionários são divididos em quatro casos de usos (Buscar, Alterar Cadastrar e Excluir). Para estes casos de usos, codificamos primeiro o Buscar, depois o Cadastrar, Excluir e por último Alterar.
6. Testes. À medida que os Casos de usos eram terminados, o teste de validação era verificado. Para que o Caso de uso fosse homologado, os analistas responsáveis pelo projeto faziam todos os testes possíveis.

5.4. Fase Transição

A fase de transição é a fase que busca como objetivo principal a implantação efetiva do sistema que foi desenvolvido. Como este trabalho visa uma pesquisa acadêmica, o software de gerenciamento de projetos proposto não será implantado e esta fase não será aplicada nesta pesquisa.

Cap. VI – Conclusão

Com os estudos realizados durante o desenvolvimento deste trabalho, pode-se constatar que com a adoção do RUP (*Rational Unified Process*) como metodologia de desenvolvimento de sistemas pode-se obter muitos benefícios, tais como: maior qualidade de software e controle no desenvolvimento, possibilitando a estimativa de prazos e custos com maior precisão.

Porém, apesar dos benefícios que esta metodologia traz para o desenvolvimento de sistemas deve-se, antes fazer uma análise para levantar se é realmente necessária à utilização de tal metodologia no projeto em questão.

O RUP é uma metodologia muito complexa, possui inúmeros documentos e, para que funcione adequadamente, é necessário seguir os quesitos das etapas do ciclo de vida do projeto, assim como também, gerar corretamente os artefatos de cada etapa. Além disso, é necessário também um treinamento adequado da equipe para adaptação e assimilação da metodologia no contexto da empresa, já que a metodologia utiliza, e muito, UML (*Unified Modeling Language*) para gerar os principais diagramas nas suas fases, principalmente na Concepção e Elaboração.

Para projetos de pequeno porte, a utilização do RUP pode não ser a melhor escolha, devido à complexidade identificada nos seus ciclos iterativos.

Já para projetos de grande porte, a utilização do RUP é muito indicada, pois possibilita maior controle, tanto do projeto quanto dos *stakeholders*, podendo assim obter um desenvolvimento dentro dos custos, prazos e níveis de qualidade desejados.

Cada empresa possui características únicas, e isso faz com que a utilização do RUP possa ser feita de forma customizada. Em cada etapa do ciclo de vida do projeto, muitos documentos são gerados, o que pode ocasionar atrasos na entrega de projetos de baixa complexidade. Para evitar que ocorram esses atrasos, faz-se necessário a escolha de apenas alguns dos inúmeros documentos que podem ser gerados ao longo do ciclo de vida do projeto. A implantação do RUP de forma customizada pode ser feita através dos estudos a seguir:

- Concepção: fase que tem como objetivo identificar o que será construído, quais as funcionalidades que o sistema terá, entre outras. Nesta fase, indica-se a elaboração do documento de visão e casos de usos.
- Elaboração: fase onde o projeto começa a sair da análise. Nesta fase, indica-se a elaboração do documento de arquitetura de software, modelo de design, modelo de dados, guia de padrões de programação da linguagem adotada e a criação de um protótipo navegável do sistema.
- Construção: esta fase se encarrega da codificação do sistema. Pode não ser gerado nenhum documento, mas é importante definir um plano de desenvolvimento das funcionalidades a serem implementadas, ou seja, traçar um roteiro dos casos de uso que serão codificados.
- Transição: nesta fase o sistema proposto é implantado. Esta fase não foi aplicada no trabalho em questão, mas como o sistema já está testado e pronto, a geração de documentos não se faz necessário. O que pode ser interessante é a criação de um plano de treinamento dos usuários, mas isso fica a critério do cliente.

Como se pode ver, o RUP é uma metodologia muito forte na questão de controle, sua adoção influencia muito do desenvolvimento de *softwares*. A customização da metodologia entra

como alternativa para qualquer tipo de projeto, evitando que estes tenham problemas com prazos e custos.

Cap. VII - Referências, Anexos

7.1. Referências Bibliográficas

AMS. *Project Management Software | Resource Management Software | Time Recording Software from AMS.* Disponível em: <http://www.amsusa.com/>. Acesso em: 27 fev. 2007.

CHONOLES, Michael Jesse; SCHARDT, James A.. *UML 2 for Dummies*. 1 ed. Indianapolis, Ed Wiley Publishing, Inc, 2003.

DOTPROJECT. *Dotproject - Open Source Software :: Open Source Project and Task Management Software :: Open Source Project and Task Management Software.* Disponível em: < http://www.dotproject.net>. Acesso em: 27 fev. 2007.

FORTES, Débora. Web 2.0: A nova cara do software. *Info Exame*, São Paulo, n. 243, p.44-63, jun. 2006. Mensal.

GANTTPROJECT. *GanttProject*. Disponível em: < http://ganttproject.sourceforge.net>. Acesso em: 27 fev. 2007.

GFORGE. *GForge CDE*. Disponível em: <http://www.gforge.org>. Acesso em: 27 fev. 2007.

HOUAISS, Antonio. *Dicionário Houaiss da Língua Portuguesa.* 1 ed. São Paulo. Objetiva.

IMENDIO. *Imendio Developer Pages.* Disponível em: <http://developer.imendio.com/>. Acesso em: 27 fev. 2007.

JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. *UML*: Guia do Usuário. 2. ed. Rio de Janeiro: Elsevier, 2005. 474 p.

KROLL, Per; KRUCHTEN, Philippe. *The Rational Unified Process Made Easy:* A Practitioner's Guide to the RUP .1 ed. Boston, Addison Wesley, 2003.

KRUCHTEN, Philippe. *Introdução ao RUP – Rational Unified Process.* 2 ed. Rio de Janeiro, Editora Ciência Moderna Ltda, 2003.

MICROSOFT. *Project Homepage.* Disponível em: <http://office.microsoft.com/pt-br/project/default.aspx>. Acesso em: 27 fev. 2007.

PMBOK. Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos: (Guia PMBOK®). 3. ed. Eua: Four Campus Boulevard, 2004.

PRADO, Darci. *Gerenciamento de Projetos nas Organizações.* Belo Horizonte: Dg, 2003. 199 p.

PROJECT-OPEN. *]project-open[*. Disponível em: <http://www.project-open.com/>. Acesso em: 27 fev. 2007.

PRESSMAN, Roger S. *Engenharia de Software*, São Paulo, Pearson Makron Books, 1995.

RATIONAL SOFTWARE CORPORATION. *Rational Unified Process.* Disponível em: <http://www.wthreex.com/rup/>. Acesso em: 20 mar. 2007.

SILVEIRA NETO, Fernando Henrique da; DISNMORE, Paul Campbell. Como gerenciar equipes de projetos e consquistar resultados através de pessoas. *Mundo PM*: Project-Program-Portfolio Management, Rio de Janeiro, n. 10, p.58-60, ago/set. 2006. Bimestral.

SOMMERVILLE, Ian. *Engenharia de Software*, 6 ed. São Paulo, Pearson Education, 2005.

VARGAS, Ricardo Viana. Identificando e Recuperando Projetos Problemáticos: Como Resgatar seu Projeto do Fracasso. *Mundo PM*: Project-Program-Portfolio Management, Rio de Janeiro, n. 10, p.42-49, ago/set. 2006. Bimestral.

7.2. Bibliografia Complementar

AHMED, Khawar Zaman; UMRYSH, Cary E.. *Desenvolvendo aplicações comerciais em Java com J2EE e UML.* Rio de Janeiro: Moderna, 2002.

ALMEIDA, Martinho Isnard Ribeiro de. *Manual de Planejamento Estratégico:* Desenvolvimento de um Plano Estratégico com a Utilização de Planilhas Excel. 2. ed. São Paulo: Atlas, 2003, 156p.

HELDMAN, Kim. *Gerência de Projetos:* fundamentos: um guia prático para quem quer certificação em gerência de projetos. Rio de Janeiro: Elsevier, 2005. 319 p.

WAZLAWICK, Raul Sidnei. *Análise e projeto de sistemas de informação orientados a objetos*. 3. ed. Rio de Janeiro: Campus, 2004.

ANEXO A – FASE CONCEPÇÃO

A1. Documento de visão

A1.1. Visão Geral do Problema

Atualmente as empresas não possuem uma ferramenta de qualidade para controlar e gerenciar seus projetos. Na maioria das vezes, os projetos fogem do escopo ou ultrapassam as datas de entregas estabelecidas, gerando assim, insatisfação dos clientes, além de acarretar muitos prejuízos à empresa executora do projeto.

A1.2. Descrição dos Envolvidos e dos Usuários

A1.2.1. Resumo dos Envolvidos

Nome	Descrição	Responsabilidades
Vinicius Miana	Auditor e Coordenador	▪ Responsável pela equipe no desenvolvimento do sistema ▪ Responsável pelo cronograma e eventuais revisões de escopo; ▪ Administração do plano e programa de trabalho ▪ Execução do cronograma
Marcos Alves	Analista de Sistemas	▪ Levantamento de requisitos segundo RUP ▪ Análise e desenho da solução segundo RUP ▪ Garantir a documentação e andamento do projeto segundo a metodologia RUP ▪ Garantir que a auditoria valide os artefatos gerados em cada fase do projeto
Rodrigo de Oliveira Neves	Analista de Sistemas	▪ Análise e desenho da solução segundo RUP ▪ Garantir a documentação e andamento do projeto segundo a metodologia RUP ▪ Garantir que a auditoria valide os artefatos gerados em cada fase do projeto
Anísio Rodrigues Neto	Analista de Sistemas	▪ Análise e desenho da solução segundo RUP ▪ Garantir a documentação e andamento do

		projeto segundo a metodologia RUP ▪ Garantir que a auditoria valide os artefatos gerados em cada fase do projeto

A1.2.2. Resumo dos Usuários

Nome	**Descrição**	**Responsabilidades**
Diretor	Diretor da Empresa	Monitora o andamento dos projetos em que a empresa está envolvida (UC14).
Gerente de Projetos	Gerente do projeto	Monitora o andamento do seu projeto (UC10). Estima as horas de cada tarefa relacionada ao seu projeto (UC05 e UC06). Assegura o cumprimento dos prazos e metas estipuladas nas tarefas (UC12). Relaciona os recursos disponíveis para as tarefas (UC03).
Usuário	Usuário do projeto	Verifica junto ao cliente as suas necessidades (UC11, UC12 e UC13). Identificação dos requisitos do projeto (UC11, UC12 e UC13). Propõem soluções aos requisitos levantados (UC11, UC12 e UC13).

A1.3. Lista de Casos de Uso identificados (UC)

UC01 – Manter Funcionário

UC02 - Vincular Permissões

UC03 - Vincular Funcionários

UC04 – Manter Tarefas

UC05 - Manter Projetos

UC06 - Manter Empresa

UC07 - Realizar Login

UC08 - Verificar Permissões

UC09 - Gerar Log de Acesso

UC10 - Consultar Andamento do Projeto

UC11 – Manter Andamento da Tarefa

UC12 - Visualizar Tarefas

UC13 - Visualizar Andamento de Tarefas

UC14 - Exibir dados dos projetos em andamento

A1.4. Requisitos Funcionais (RF)

RF1 – Controlar Projetos

RF2 – Controlar Tarefas

RF3 – Calcular porcentagens de andamento dos projetos

RF4 – Registrar Logs de Tarefas

RF5 – Registrar Logs de Acesso

RF6 – Controlar Acessos

RF7 – Calcular porcentagens de andamento das tarefas

RF8 – Controlar Funcionários

RF9 – Controlar Clientes

A1.5. Requisitos Não Funcionais (RNF)

Disponibilidade

As funcionalidades não devem afetar as operações do servidor de aplicações e deve estar disponível por 24 horas, durante os 7 dias da semana.

Segurança

Os usuários do sistema devem acessar apenas as funcionalidades que foram vinculadas em seu cadastro. As funcionalidades são módulos de acesso ao sistema.

A1.6. Visão Geral do Sistema Proposto

A1.6.1. Características do Sistema Proposto

Este sistema terá como finalidade gerenciar todos os projetos cadastrados, assim como as suas tarefas relacionadas. O objetivo do sistema é controlar de forma detalhada o andamento de cada projeto, além do cumprimento de suas tarefas. O sistema deverá calcular automaticamente as porcentagens de conclusão de cada projeto a partir dos logs das tarefas gerados pelos usuários, que podem ser os analistas, desenvolvedores e/ou projetistas. Deverá possuir consultas e relatórios que auxiliem os Diretores acompanharem o andamento dos projetos da sua empresa, assim como os Gerentes de Projetos terem acesso às informações referentes ao seu projeto. O sistema deverá ter controle de acesso, onde cada usuário terá permissão específica às funcionalidades, além disso, proporcionar o controle de logs das ações de cada usuário conectado no sistema.

A1.6.2. Ambiente Alvo

Tecnologia	**Padrão**	**Versão**	**OBS.**
Servidor de Banco de Dados			
	MySql	5.0	
Linguagem de Programação			
	JAVA, JavaScript, XML, Ajax		

A2. Casos de uso

A2.1. Atores

- **Administrador do Sistema.** Este ator possui acesso total à todas funcionalidades do sistema.
- **Gerente de Projetos.** Responsável no acompanhamento do projeto que coordena, assim como, vincular as tarefas para a equipe do seu projeto.
- **Usuário.** Ator que alimenta a maior parte do sistema. Será ele que controlará sua tarefa, assim como, o status (log) desta.
- **Diretor.** Diretor da empresa que terá a visão macro dos andamentos dos projetos em que a sua empresa esta executando.

A2.2. Lista de Casos de Usos

UC01 – Manter Funcionário

UC02 - Vincular Permissões

UC03 - Vincular Funcionários

UC04 – Manter Tarefas

UC05 - Manter Projetos

UC06 - Manter Empresa

UC07 - Realizar Login

UC08 - Verificar Permissões

UC09 - Gerar Log de Acesso

UC10 - Consultar Andamento do Projeto

UC11 – Manter Andamento da Tarefa

UC12 - Visualizar Tarefas

UC13 - Visualizar Andamento de Tarefas

UC14 - Exibir dados dos projetos em andamento

A2.3. Casos de Usos

ud Use Case Model

Administrador

UC01 - Manter Funcionário

UC09 - Gerar Log de Acesso

«include»

«extend»

UC02 - Vincular Permissões

«include»

UC04 - Manter Tarefas

UC09 - Gerar Log de Acesso

«include»

«extend»

«include»

UC03 - Vincular Funcionários

UC05 - Manter Projetos

«include»

UC09 - Gerar Log de Acesso

UC06 - Manter Empresa

«include»

UC09 - Gerar Log de Acesso

UC11 - Manter Logs de Tarefas

«include»

UC08 - Verificar Permissões

UC07 - Realizar Login

«include»

A2.4. Especificação dos Casos de Usos

A2.4.1. UC01 – Manter Funcionário

Breve Descrição

Este caso de uso tem como finalidade cadastrar, alterar ou excluir funcionários que utilizarão o sistema de gerenciamento de projetos.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o administrador vai cadastrar alterar ou excluir um funcionário.

Cadastrar Funcionário

O administrador deverá informar os seguintes dados do funcionário em um formulário:

1) Nome *****
2) Departamento *****
3) CPF *****
4) RG *****
5) Senha *****
6) E-mail
7) Rua *****
8) Número da casa *****
9) Complemento do endereço
10) Bairro *****
11) Cidade *****
12) Estado *****
13) Pais *****
14) CEP *****
15) DDD telefone *****
16) Número do Telefone *****
17) Data de nascimento *****
18) Tipo funcionário *

** Campos obrigatórios*

Depois disso, deverá clicar no botão Incluir. Após a inclusão do funcionário, o administrador pode escolher ou não vincular as permissões de acesso ao sistema do funcionário cadastrado, executando o caso de uso UC02 – Vincular Permissões.

OBS.: As permissões de acesso encontram-se cadastradas na tabela PERMISSAO.

Os tipos de funcionário são:

1 – Administrador;

2 – Gerente de Projetos;

3 – Analista, programador, projetista, etc.

Alterar Dados dos Funcionários

O administrador deverá selecionar o usuário que deseja alterar. Automaticamente os dados daquele funcionário serão carregados na tela. O administrador alterará os dados necessários e clicará no botão Alterar. O único dado que não poderá ser alterado é o CPF. Após a alteração dos dados do funcionário, o administrador pode escolher ou não alterar as permissões de acesso ao sistema daquele funcionário alterado, executando o caso de uso UC02 – Vincular Permissões.

OBS.: As permissões de acesso encontram-se cadastradas na tabela PERMISSAO.

Desativar Funcionário

O administrador deverá selecionar o usuário que deseja excluir e clicar no botão Excluir para que os dados daquele funcionário possam ser desativados no sistema. Além de desativar o funcionário, as permissões de acesso ao sistema do mesmo, devem ser excluídas da tabela PERMISSAO.

Fluxos alternativos

Funcionário já cadastrado

Este fluxo ocorre quando o administrador tenta inserir um funcionário com o CPF igual ao que já se encontra cadastrado no banco de dados.

Emitir mensagem: “Funcionário já cadastrado”.

Dados Incompletos

Este fluxo ocorre quando algum campo obrigatório não foi preenchido no formulário.

Emitir mensagem dos campos que estão faltando e colocar o foco no primeiro campo incorreto.

Funcionário Não Selecionado

Este fluxo ocorre quando não foi selecionado nenhum funcionário para as opções de alterar ou excluir.

Emitir mensagem “Selecione um funcionário para alterar ou desativar”.

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como administrador.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.2. UC02 – Vincular Permissões

Breve Descrição

Este caso de uso tem como finalidade vincular as permissões de acesso ao sistema para os funcionários cadastrados.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o administrador ao cadastrar ou alterar um funcionário, terá que vincular suas permissões de acesso ao sistema. O administrador selecionará ou alterará as permissões para o funcionário e depois clicar em Gravar Permissões. Este caso de uso carregará uma tela com todas as permissões de acesso ao sistema. Se for vincular as permissões de um funcionário novo, deverá carregar todas as permissões com nenhuma selecionada, caso seja de um funcionário que seus dados foram alterados, deverá carregar todas as permissões, mas aquelas que já fora ativadas no primeiro cadastro devem estar selecionadas.

OBS.: As permissões de acesso encontram-se cadastradas na tabela PERMISSAO.

As permissões são listadas a seguir:

- incluir/ alterar/ excluir projetos
- incluir/ alterar/ excluir tarefas
- consultar andamento dos projetos
- visualizar tarefas
- incluir/ alterar/ excluir empresas
- incluir/ alterar/ desativar funcionários
- acessar o sistema
- incluir/ alterar/ excluir andamento de tarefas

Fluxos alternativos

Nenhum

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como administrador.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.3. UC03 – Vincular Funcionários

Breve Descrição

Este caso de uso tem como finalidade vincular os funcionários às tarefas cadastradas.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o administrador ou gerente de projetos ao cadastrar ou alterar uma tarefa, terá que vincular os funcionários que participarão da mesma. O administrador ou gerente de projetos selecionará ou alterará os funcionários e depois clicar em Gravar Alterações. Este caso de uso carregará uma tela com todos os funcionários cadastrados no sistema. Se for vincular funcionários para uma nova tarefa, o sistema deverá carregar todos os funcionários com nenhum selecionado, caso seja de uma tarefa que seus dados foram alterados, deverá carregar todos os funcionários, mas aquelas que já fora ativados no primeiro cadastro devem estar selecionados.

Fluxos alternativos

Nenhum

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como administrador ou gerente de projetos.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.4. UC04 – Manter Tarefas

Breve Descrição

Este caso de uso tem como finalidade cadastrar, alterar ou excluir Tarefas dos projetos do sistema proposto.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o administrador ou o Gerente do Projeto vai cadastrar, alterar ou excluir uma tarefa de um determinado projeto.

Cadastrar Tarefas

O administrador ou gerente de projetos deverá informar os seguintes dados em um formulário:

1) Projeto *
2) Nome da tarefa*
3) Status atual * (Ativo/Inativo)
4) Prioridade *
5) Horas estimadas
6) Descrição
7) Data de início *
8) Data final *

***** *Campos obrigatórios*

Depois disso, deverá clicar no botão Incluir. Após a inclusão da tarefa, o administrador ou gerente do projeto pode escolher ou não vincular um ou mais funcionários à tarefa cadastrada, executando o caso de uso UC03 – Vincular Funcionários.

Alterar Tarefas

O administrador ou gerente de projetos deverá selecionar a tarefa que deseja alterar. Automaticamente os dados daquela tarefa serão carregados na tela. O administrador alterará os dados necessários e clicará no botão Alterar. Depois que os dados da tarefa forem alterados, o administrador ou gerente do projeto poderá alterar também os funcionários vinculados à esta tarefa, executando o caso de uso UC03 – Vincular Funcionários.

Excluir Tarefas

O administrador ou gerente de projetos deverá selecionar a tarefa que deseja excluir e clicar no botão Excluir para que os dados daquela tarefa selecionada possam ser excluídos do sistema. Para que a exclusão seja efetuada é necessário que não tenha nenhum log de andamento de tarefa cadastrado da tarefa selecionada ou nenhum funcionário vinculado à tarefa.

Fluxos alternativos

Tarefa já cadastrada

Este fluxo ocorre quando existe no banco de dados uma tarefa já cadastrada com o mesmo nome da tarefa que se deseja cadastrar.

Emitir mensagem: “Esta tarefa já está cadastrada”.

Dados Incompletos

Este fluxo ocorre quando algum campo obrigatório não foi preenchido no formulário.

Emitir mensagem dos dados que estão faltando e colocar o foco no primeiro campo incorreto.

Tarefa Não Selecionada

Este fluxo ocorre quando não foi selecionada nenhuma tarefa para as opções de alterar ou excluir.

Emitir mensagem “Selecione uma tarefa para alterar ou excluir”.

Tarefa com andamento cadastrado

Este fluxo ocorre quando existe um ou mais registros de andamento de tarefa para a tarefa que se deseja excluir.

Emitir mensagem “Não é possível excluir a tarefa selecionada, pois existem andamento de tarefas cadastrados”.

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como administrador ou gerente de projetos.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.5. UC05 – Manter Projetos

Breve Descrição

Este caso de uso tem como finalidade cadastrar, alterar ou excluir Projetos do sistema proposto.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o administrador ou o Gerente do Projeto vai cadastrar alterar ou excluir um projeto no sistema.

Cadastrar Projetos

O Gerente de Projetos deverá informar os seguintes dados no formulário:

1) Empresa *****

2) Nome do projeto*****

3) Descrição

4) Ger. Projeto responsável *****

5) Data de inicio *****

6) Data final *****

7) Prioridade

8) Status ***** (Ativo/ Inativo)

9) Orçamento

***** *Campos obrigatórios*

Depois disso, deverá clicar no botão Incluir.

Alterar Projetos

O administrador ou Gerente do Projeto deverá selecionar o projeto que deseja alterar. Automaticamente os dados daquele projeto serão carregados na tela. O administrador ou gerente do projeto alterará os dados necessários e clicará no botão Alterar.

Excluir Projetos

O administrador deverá selecionar o projeto que deseja excluir e clicar no botão Excluir para que os dados daquele projeto selecionado possam ser excluídos do sistema. Para que a exclusão seja efetuada é necessário que não tenha **nenhuma tarefa** cadastrada no projeto selecionado.

Fluxos alternativos

Projeto já cadastrado

Este fluxo ocorre quando existe no banco de dados um projeto já cadastrado com o mesmo nome do projeto que se deseja cadastrar.

Emitir mensagem: “Este projeto já está cadastrado”.

Dados Incompletos

Este fluxo ocorre quando algum campo obrigatório não foi preenchido no formulário.

Emitir mensagem dos dados que estão faltando e colocar o foco no primeiro campo incorreto.

Projeto Não Selecionado

Este fluxo ocorre quando não foi selecionado nenhum projeto para as opções de alterar ou excluir.

Emitir mensagem “Selecione um projeto para alterar ou excluir”.

Projeto com tarefas cadastradas

Este fluxo ocorre quando existe um ou mais registros de tarefas para o projeto que se deseja excluir.

Emitir mensagem “Não é possível excluir o projeto selecionado, pois existem tarefas cadastradas”.

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como administrador ou gerente de projetos.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.6. UC06 – Manter Empresa

Breve Descrição

Este caso de uso tem como finalidade cadastrar, alterar ou excluir empresas no sistema proposto.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o administrador vai cadastrar alterar ou excluir uma empresa do sistema.

Cadastrar Empresa

O administrador deverá informar os seguintes dados em um formulário:

1) Nome da empresa *
2) E-mail *
3) Descrição
4) Site *
5) Responsável
6) CNPJ *
7) Rua *
8) Número da empresa*
9) Complemento do endereço
10) Bairro *
11) Cidade *
12) Estado *
13) Pais *
14) CEP *
15) DDD telefone *
16) Número do telefone *

** Campos obrigatórios*

Depois disso, deverá clicar no botão Incluir.

Alterar Empresa

O administrador deverá selecionar a empresa que deseja alterar. Automaticamente os dados daquela empresa serão carregados na tela. O administrador alterará os dados necessários e clicará no botão Alterar.

Excluir Empresa

O administrador deverá selecionar a empresa que deseja excluir e clicar no botão Excluir para que os dados daquela empresa selecionada possam ser excluídos do sistema. Para que a exclusão seja efetuada é necessário que não tenha **nenhum projeto** cadastrado com a empresa selecionada.

Fluxos alternativos

Empresa já cadastrada

Este fluxo ocorre quando existe no banco de dados uma empresa já cadastrada com o mesmo CNPJ da empresa que se deseja cadastrar.

Emitir mensagem: “Esta empresa já está cadastrada”.

Dados Incompletos

Este fluxo ocorre quando algum campo obrigatório não foi preenchido no formulário.

Emitir mensagem dos dados que estão faltando e colocar o foco no primeiro campo incorreto.

Empresa Não Selecionada

Este fluxo ocorre quando não foi selecionada nenhuma empresa para as opções de alterar ou excluir.

Emitir mensagem “Selecione uma empresa para alterar ou excluir”.

Empresa com projetos cadastrados

Este fluxo ocorre quando existe um ou mais registros de projetos para a empresa que se deseja excluir.

Emitir mensagem “Não é possível excluir a empresa selecionada, pois existem projetos vinculados”.

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como administrador.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.7. UC07 – Realizar Login

Breve Descrição

Este caso de uso tem como finalidade dar acesso ao sistema.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando qualquer usuário em geral, quer acessar o sistema.

O usuário acessa a página de inicial do sistema. Fornece seus dados de login * e senha * e clica em Logar. O sistema verificará se o usuário é cadastrado e se o mesmo possui permissão para acessar o sistema executando o caso de uso UC08 – Verificar Permissões. Após isto, o usuário terá acesso à página principal do sistema. Após verificar as permissões e montar o menu de opções na tela, o sistema buscará todas as tarefas do usuário em questão executando o UC12 – Visualizar Tarefas.

** Campos obrigatórios*

Fluxos alternativos

Login ou senha inválidos

Este fluxo ocorre quando o usuário digita a senha ou o login incorretamente.

Emitir mensagem: "Login ou senha inválido".

Dados Incompletos

Este fluxo ocorre quando algum campo obrigatório não foi preenchido no formulário.

Emitir mensagem dos dados que estão faltando e colocar o foco no primeiro campo incorreto.

Sem permissão

Este fluxo ocorre quando o usuário não tem permissão para acessar o sistema.

Emitir mensagem "Você não possui permissão para acessar o sistema".

Condições Prévias

Nenhuma

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

A2.4.8. UC08 – Verificar Permissões

Breve Descrição

Este caso de uso tem como finalidade dar acesso ao sistema de acordo com as permissões cadastradas de um determinado usuário.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando qualquer usuário em geral, quer acessar o sistema.

Após validar login e senha, este caso de uso entra em atividade verificando as permissões de acesso, colocando na página principal somente o que o usuário logado tem acesso.

Fluxos alternativos

Nenhum

Condições Prévias

Ter login e senha validados

O usuário deverá ter login e senha validado pelo sistema.

Condições Posteriores

Nenhuma.

A2.4.9. UC09 – Gerar Log de Acesso

Breve Descrição

Este caso de uso tem como finalidade gerar log de todas ações feitas pelos usuários cadastrados no sistema.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando qualquer usuário em geral, executa uma determinada funcionalidade no sistema.

Após executar qualquer funcionalidade no sistema, este caso de uso inicia-se gerando o log desta operação. O log deverá conter os seguintes dados:

1 – Código do funcionário (CPF);

2 – IP do computador do usuário;

3 – Descrição da ação;

4 – Data da geração do log.

Fluxos alternativos

Nenhum

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sitema.

Condições Posteriores

Nenhuma.

A2.4.10. UC10 – Consultar Andamento do Projeto

Breve Descrição

Este caso de uso tem como finalidade fornecer informações referente aos projetos cadastrados no sistema.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o gerente do projeto quer obter informações dos seus projetos.

O sistema mostrará uma lista contendo o nome, descrição e progresso de andamento dos projetos do gerente responsável. Após isso, o gerente terá a possibilidade de visualizar mais informações como data de inicio, data de termino, prioridade, cliente e status atual do projeto clicando em Detalhes.

Fluxos alternativos

Projeto Não Selecionado

Este fluxo ocorre quando não foi selecionado nenhum projeto para as opções de detalhes.

Emitir mensagem “Selecione um projeto”.

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como gerente de projetos.

Condições Posteriores

Nenhuma.

A2.4.11. UC11 – Manter Andamento da Tarefa

Breve Descrição

Este caso de uso tem como finalidade cadastrar, alterar ou excluir andamentos das tarefas.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o usuário vai cadastrar alterar ou excluir um andamento de uma determinada tarefa.

Cadastrar Andamento da Tarefa

O usuário deverá selecionar a tarefa que está participando e informar os seguintes dados no formulário:

1) Horas trabalhadas *****
2) Data *****
3) Descrição *****
4) Progresso de andamento da tarefa

***** *Campos obrigatórios*

Depois disso, deverá clicar no botão Incluir.

OBS.: Funcionalidade que qualquer usuário do sistema pode realizar, desde que tenha alguma tarefa vinculada ao mesmo.

Alterar Andamento da Tarefa

O administrador deverá selecionar o andamento da tarefa que deseja alterar. Os dados do andamento da tarefa serão carregados na tela. Após alterar os dados, o usuário poderá clicar no botão Alterar para que os dados possam ser alterados no sistema.

Excluir Andamento da Tarefa

O administrador deverá selecionar o andamento da tarefa que deseja excluir e clicar no botão Excluir para que os dados possam ser excluídos no sistema.

Fluxos alternativos

Dados Incompletos

Este fluxo ocorre quando algum campo obrigatório não foi preenchido no formulário.

Emitir mensagem dos dados que estão faltando e colocar o foco no primeiro campo incorreto.

Andamento da Tarefa Não Selecionada

Este fluxo ocorre quando não foi selecionado nenhum andamento de tarefa para a opção de excluir ou alterar.

Emitir mensagem “Selecione um andamento de tarefa para excluir ou alterar”.

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema.

Condições Posteriores

Após a execução do caso de uso, gerar log da operação, executando caso de uso UC09 – Gerar Log de Acesso.

Também deverá ser recalculada a porcentagem de conclusão da tarefa e posteriormente do projeto que o andamento foi cadastrado.

A2.4.12. UC12 – Visualizar Tarefas

Breve Descrição

Este caso de uso tem como finalidade mostrar todas as tarefas vinculadas para um determinado funcionário

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o usuário acessa o sistema.

O usuário pode através deste caso de uso visualizar todas as suas tarefas. O sistema irá capturar todas as tarefas que estão vinculadas ao usuário e apresentar uma lista contendo os seguintes dados: projeto, tarefa, data de início e data de término da tarefa. Ao apresentar na tela a lista, o usuário poderá clicar em qualquer tarefa para verificar seus andamentos, executando assim o caso de uso UC13 - Visualizar Andamento de Tarefas.

Fluxos alternativos

Nenhum

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema.

Condições Posteriores

Nenhuma

A2.4.13. UC13 – Visualizar Andamento de Tarefas

Breve Descrição

Este caso de uso tem como finalidade mostrar todos os andamentos das tarefas vinculadas para um determinado funcionário.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o usuário quer ver os andamentos de suas tarefas.

O sistema irá capturar os andamentos da tarefa e apresentar uma lista contendo o nome do usuário, descrição, data e horas trabalhadas do andamento da tarefa. Cada andamento da tarefa deverá conter um link para poder ser alterado (UC11 – Manter Andamento da Tarefa). Somente os andamentos que foram cadastrados por outro usuário da mesma tarefa, não poderão ter link para alteração.

Fluxos alternativos

Nenhum

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema.

Condições Posteriores

Nenhuma

A2.4.14. UC14 – Exibir dados dos projetos em andamento

Breve Descrição

Este caso de uso tem como finalidade mostrar dados de todos os projetos que a empresa está administrando.

Fluxo de Eventos

Fluxo Básico

Este caso de uso se inicia quando o diretor da empresa quer ver os projetos que estão em andamento.

O sistema irá capturar todos os projetos que estão em andamento e apresentar na tela uma lista contendo o nome, descrição e progresso de andamento do projeto. Após isso, o diretor terá a possibilidade de visualizar mais informações como data de início, data de término, prioridade, cliente e status atual do projeto clicando em Detalhes na lista.

Fluxos alternativos

Nenhum

Condições Prévias

Estar logado no sistema

O usuário deverá estar logado no sistema como diretor.

Condições Posteriores

Nenhuma

ANEXO B – FASE ELABORAÇÃO

B1. Documento de arquitetura de software

B1.1. Objetivo

Este documento apresenta um panorama geral do sistema, utilizando diferentes visões de arquitetura que possam modelar os diferentes aspectos do sistema. A intenção é apresentar as decisões arquiteturais feitas sobre o sistema. Também são representados diferentes níveis de abstração do projeto, bem como suas relações.

B1.2. Representação da Arquitetura

A arquitetura de software é a definição e a representação de uma estrutura para a composição do sistema, em termos de seus componentes, propriedades e interações. A arquitetura deste sistema está baseada na arquitetura de referência JEE da Sun Microsystems. Portanto a representação desta arquitetura contempla apenas aspectos particulares a este sistema, que seguem as definições da arquitetura de referência.

Estão representados os casos de uso críticos, agrupamento em pacotes e a implantação do sistema.

B1.3. Objetivos e Limitações da Arquitetura

Mecanismo de Análise (Conceitual)	Mecanismo de Projeto (Concreto)	Mecanismo de Implementação (Atual)
Persistência	Banco de dados Relacional (MySQL)	JDBC via procedures
Ambiente	Servidor de Aplicações	Weblogic 8.x ou JBoss 4.x.x
Distribuição	RMI	Java 1.4 da Sun ou superior

B1.4. Visão Lógica

A divisão de pacotes deste sistema segue a hierarquia definida na arquitetura de referência. Portanto a definição é a seguinte:

- Sistema : sgp

B1.4.1. Apresentação

Este sistema possui interface com usuário. Desta forma, teremos uma camada de apresentação, camadas de negocio, transação e persistência. A hierarquia de pacotes segue a definição abaixo.

B1.4.2. Projeto dos Pacotes

- Nome: **net.sgp.empresa**
  Descrição : Pacote responsável pelo cadastro de clientes (empresas).

- Nome: **net.sgp.colaborador**
  Descrição : Pacote responsável pela cadastro do colaborador (funcionário).

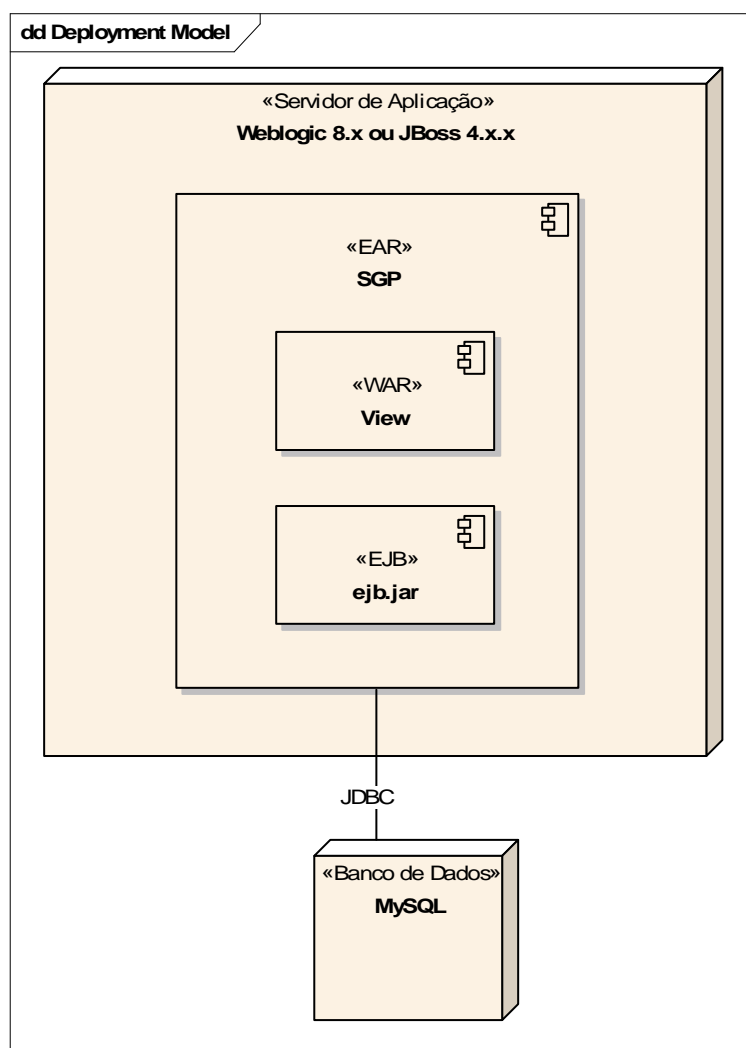
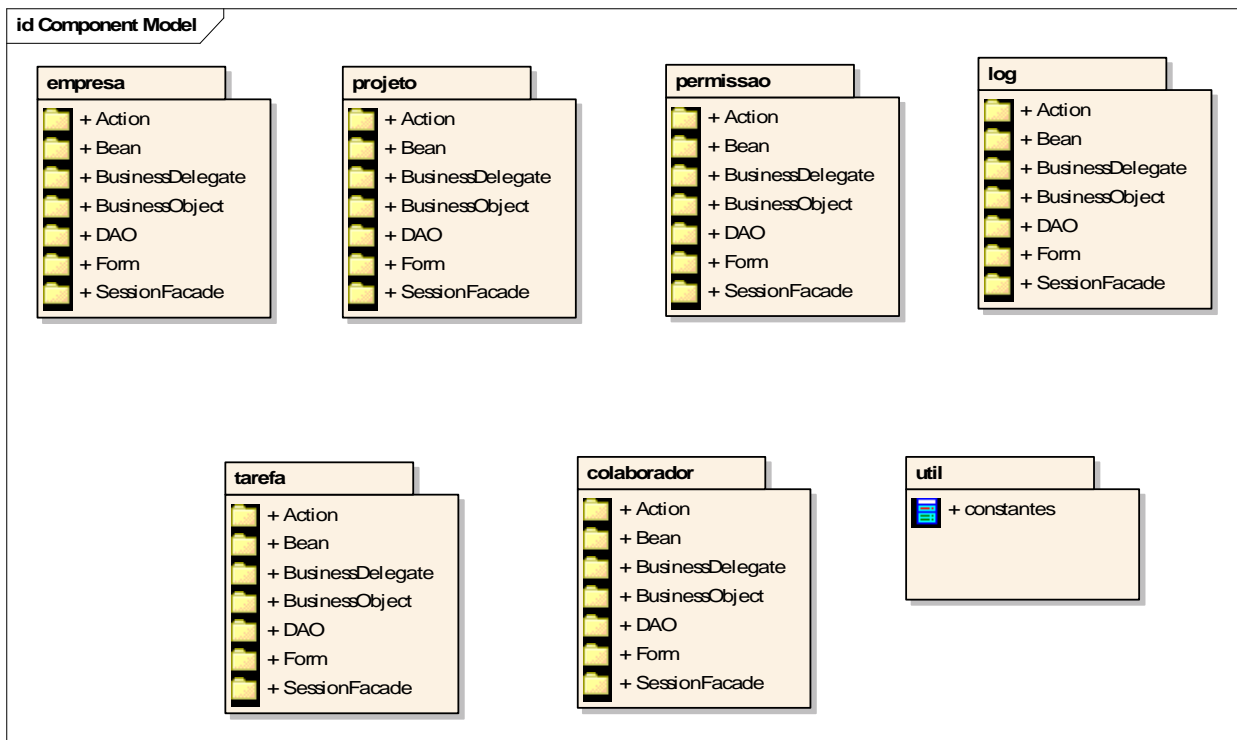
- Nome: **net.sgp.util**
  Descrição : Pacote para várias utilidade, como constantes.

- Nome: **net.sgp.tarefa**
  Descrição : Pacote responsável pelo cadastro de tarefas e andamento das tarefas.

- Nome: **net.sgp.projeto**
  Descrição : Pacote responsável pelo cadastro de projetos.

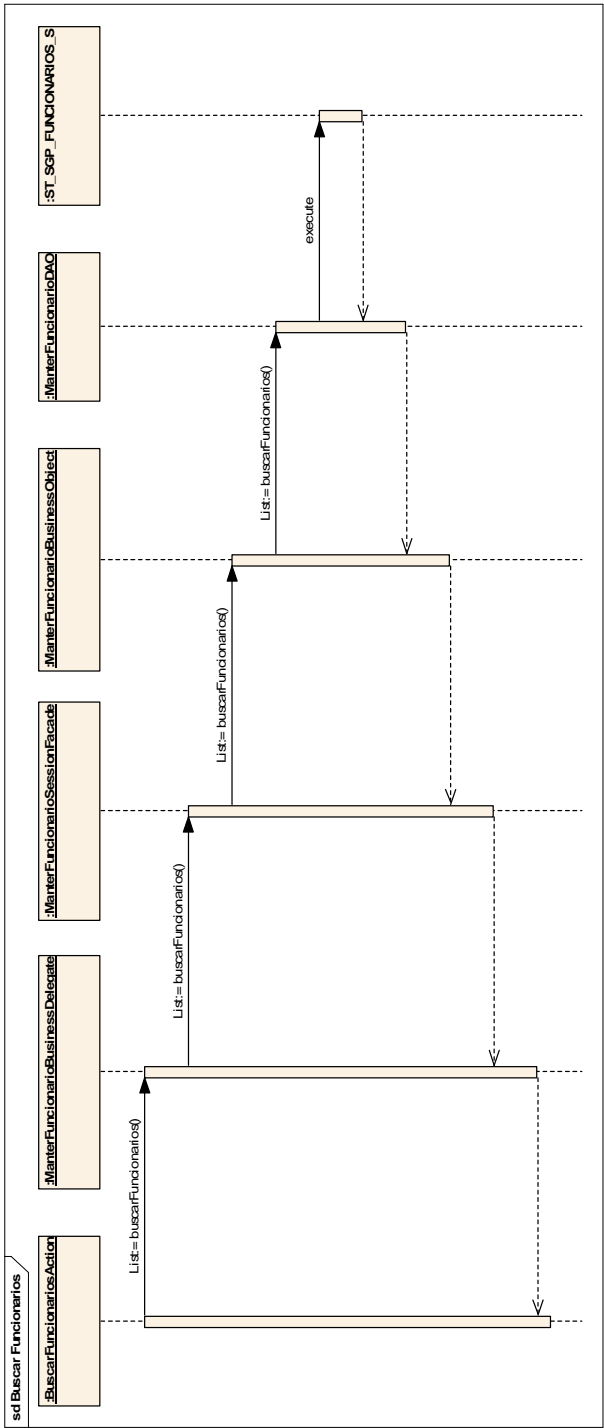
- Nome: **net.sgp.permissao**
  Descrição : Pacote responsável pelas permissões dos usuários.

- Nome: **net.sgp.log**
  Descrição : Pacote responsável pela geração de logs.

B1.5. Visão de Implantação

B1.6. Visão de Implementação

B2. Modelo de design

B2.1. Diagrama de seqüência

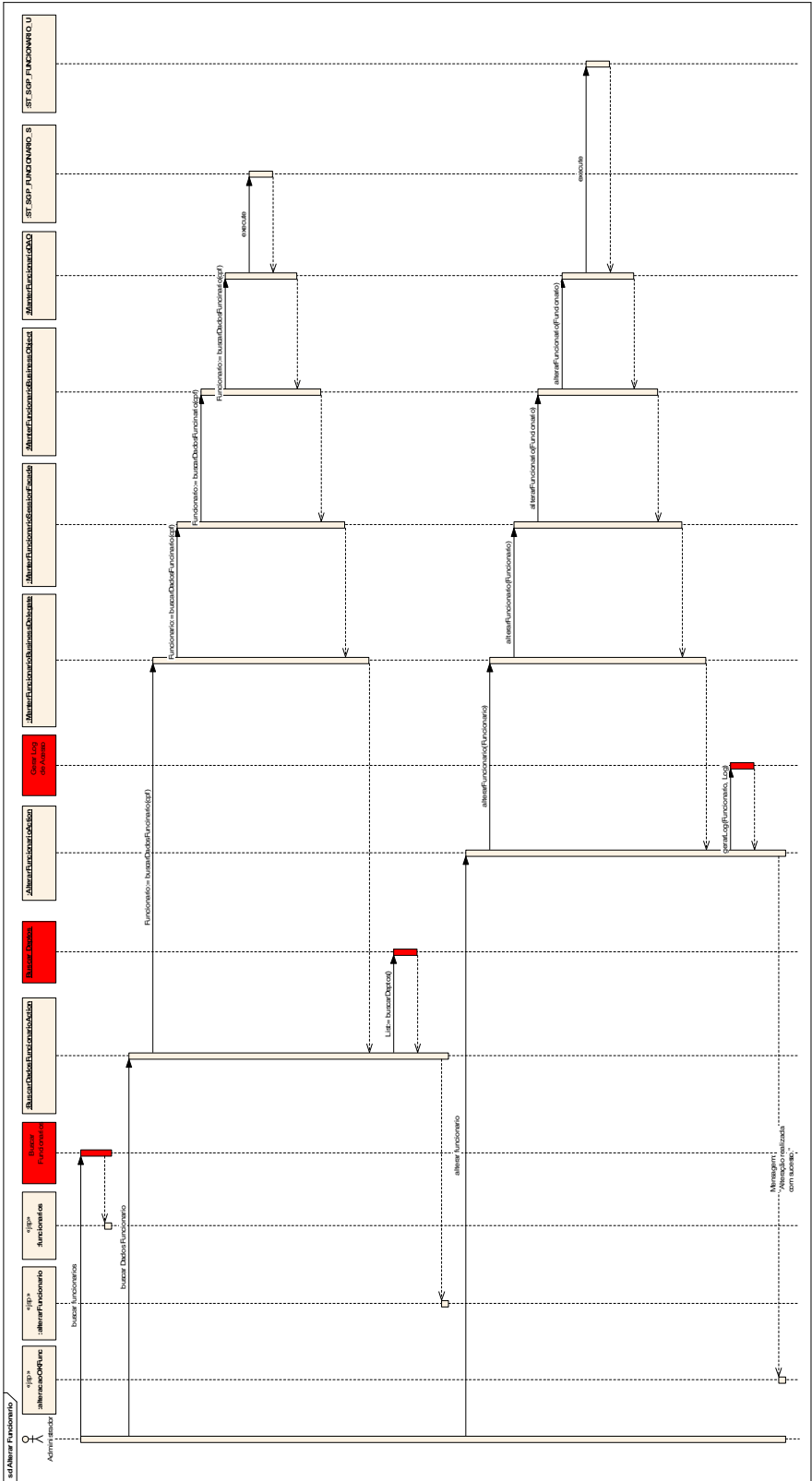
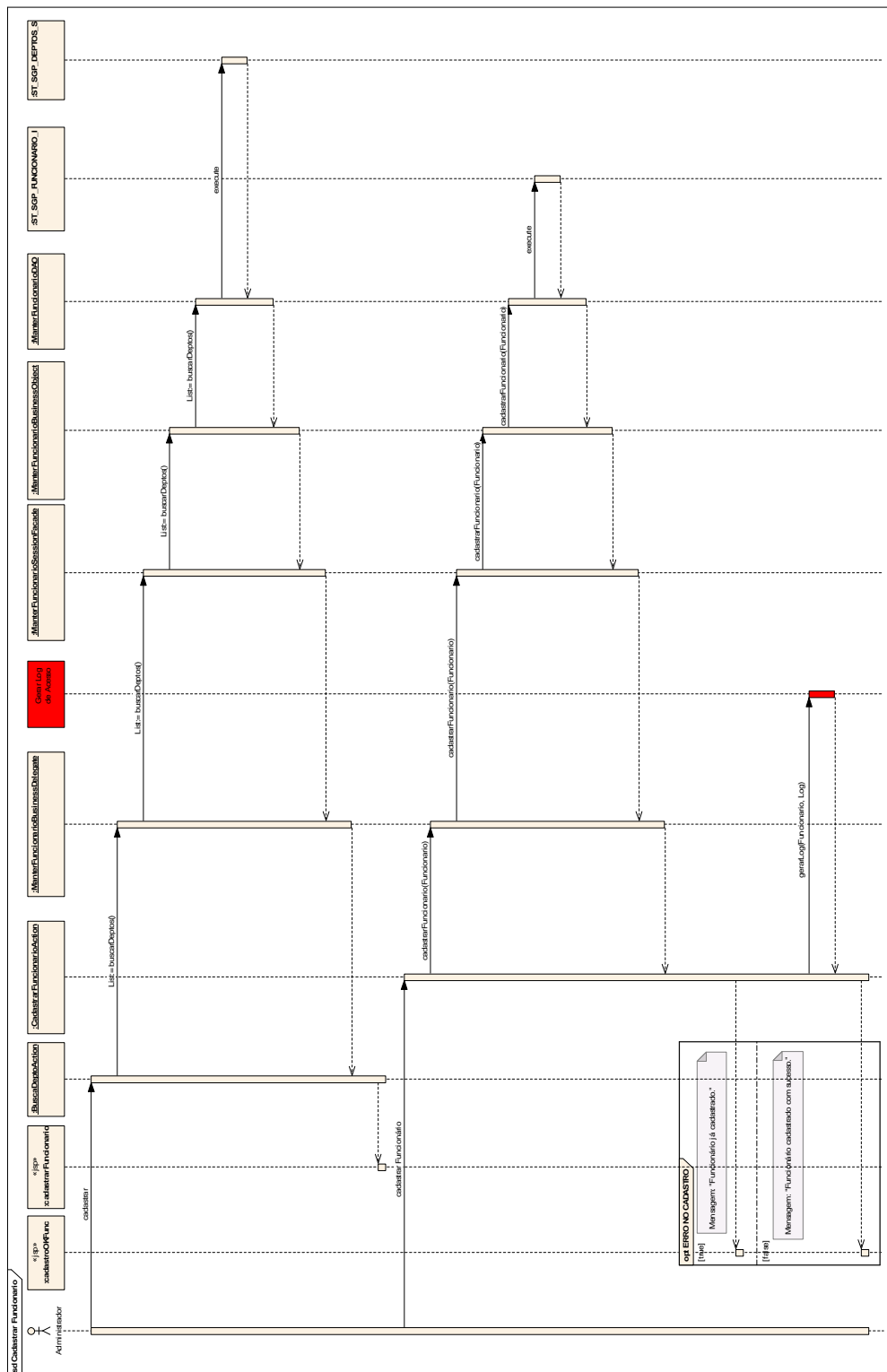
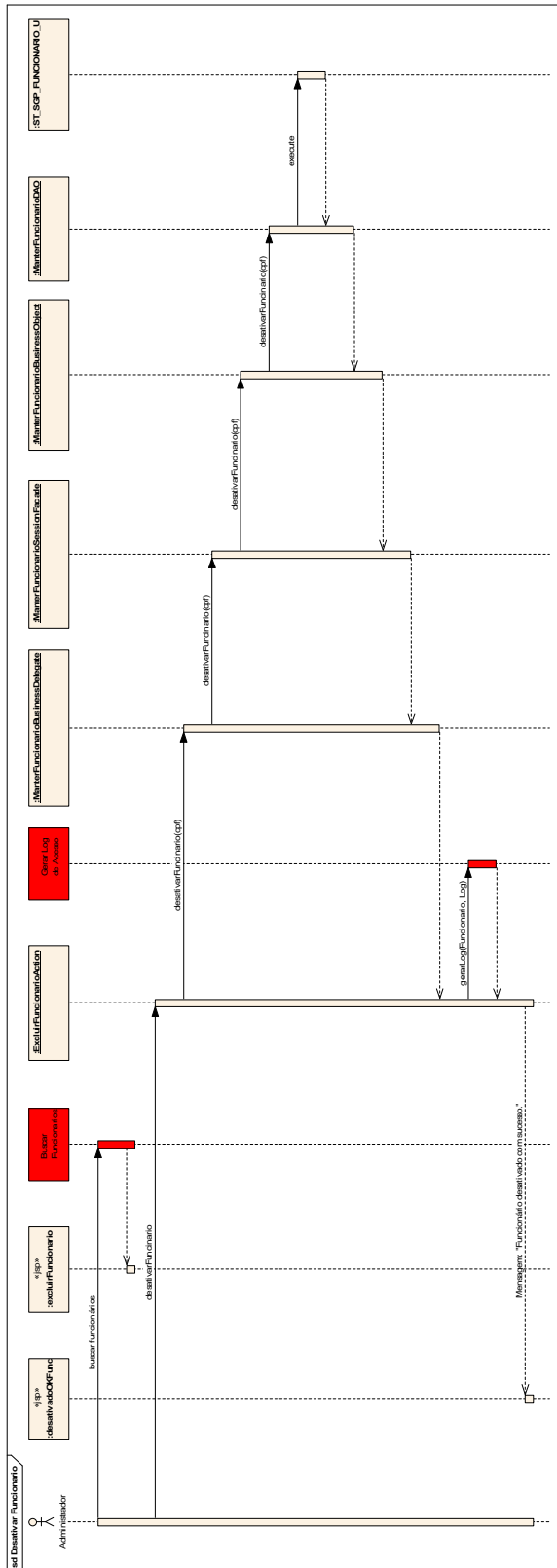
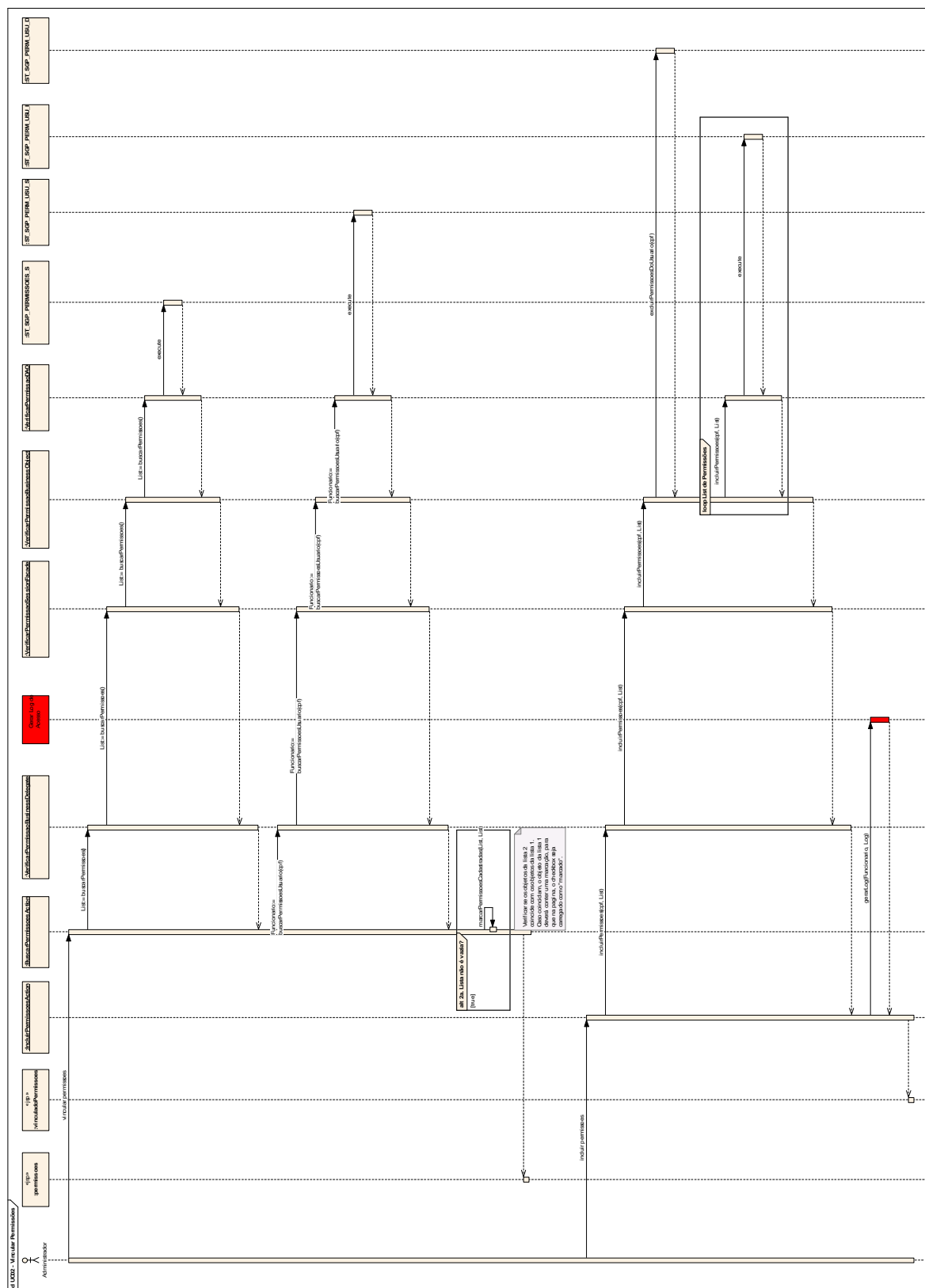
B2.1.1. UC01 – Manter Funcionário

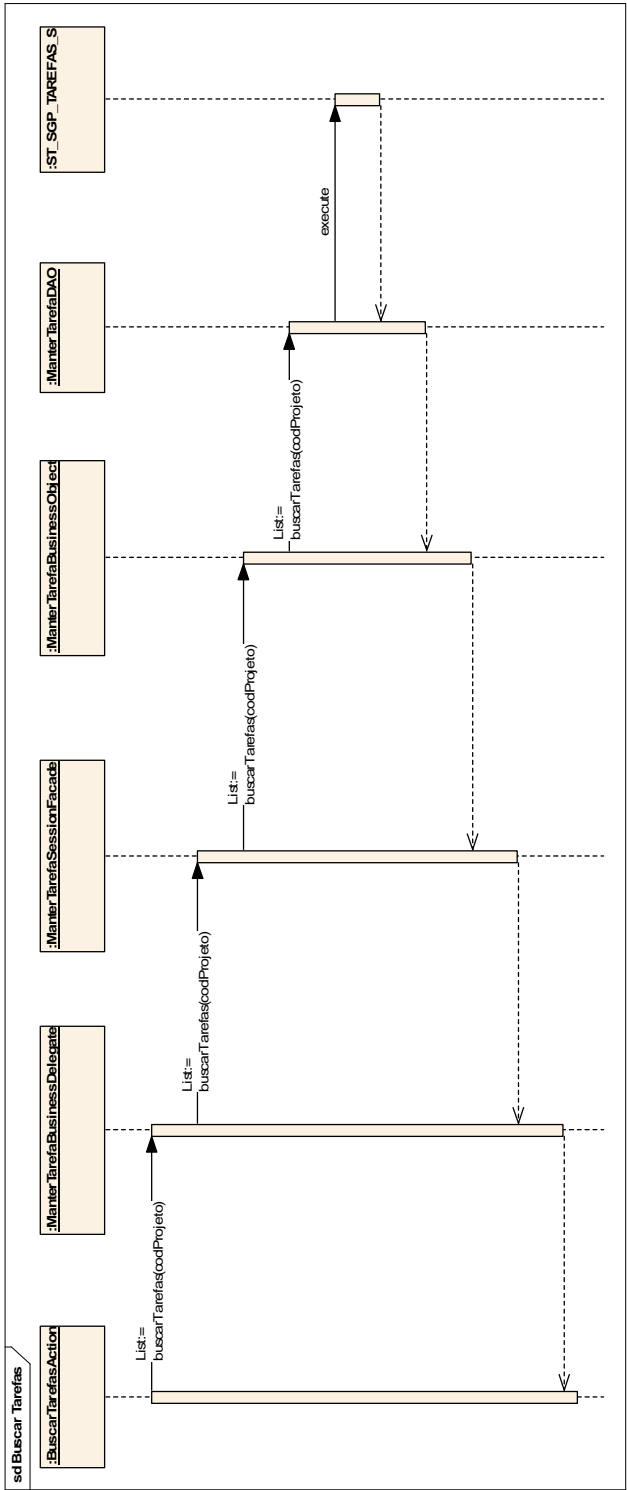
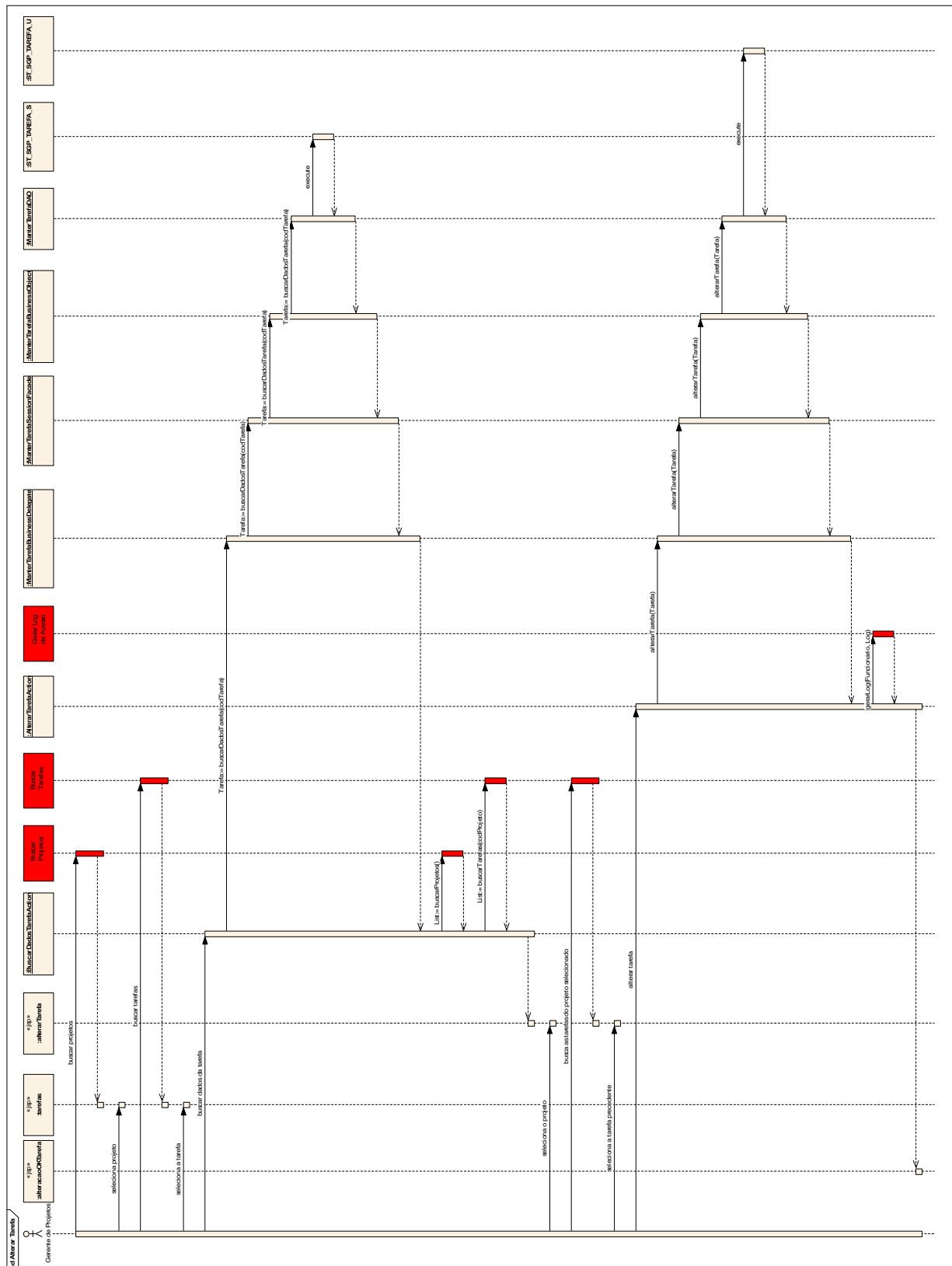
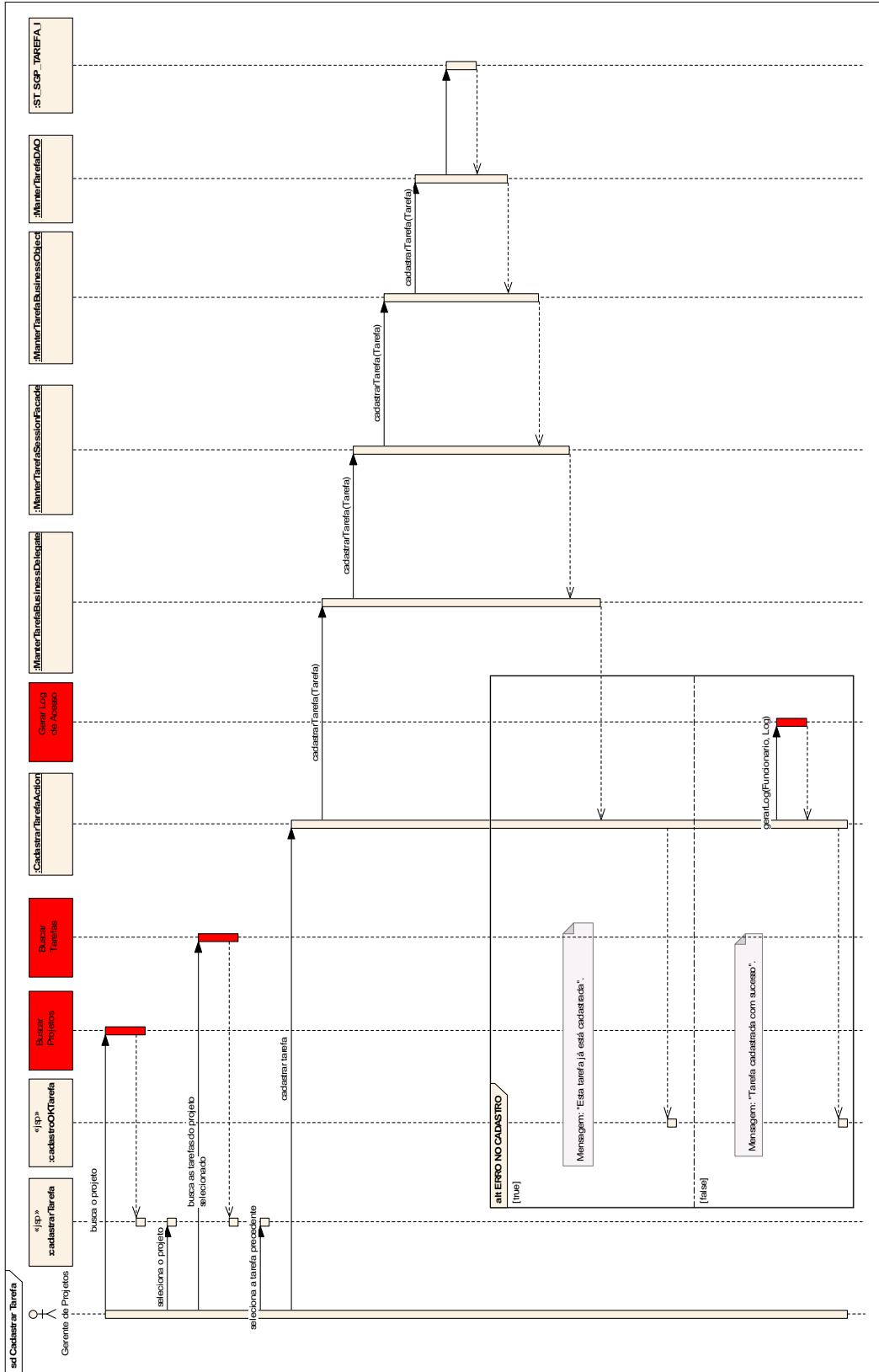
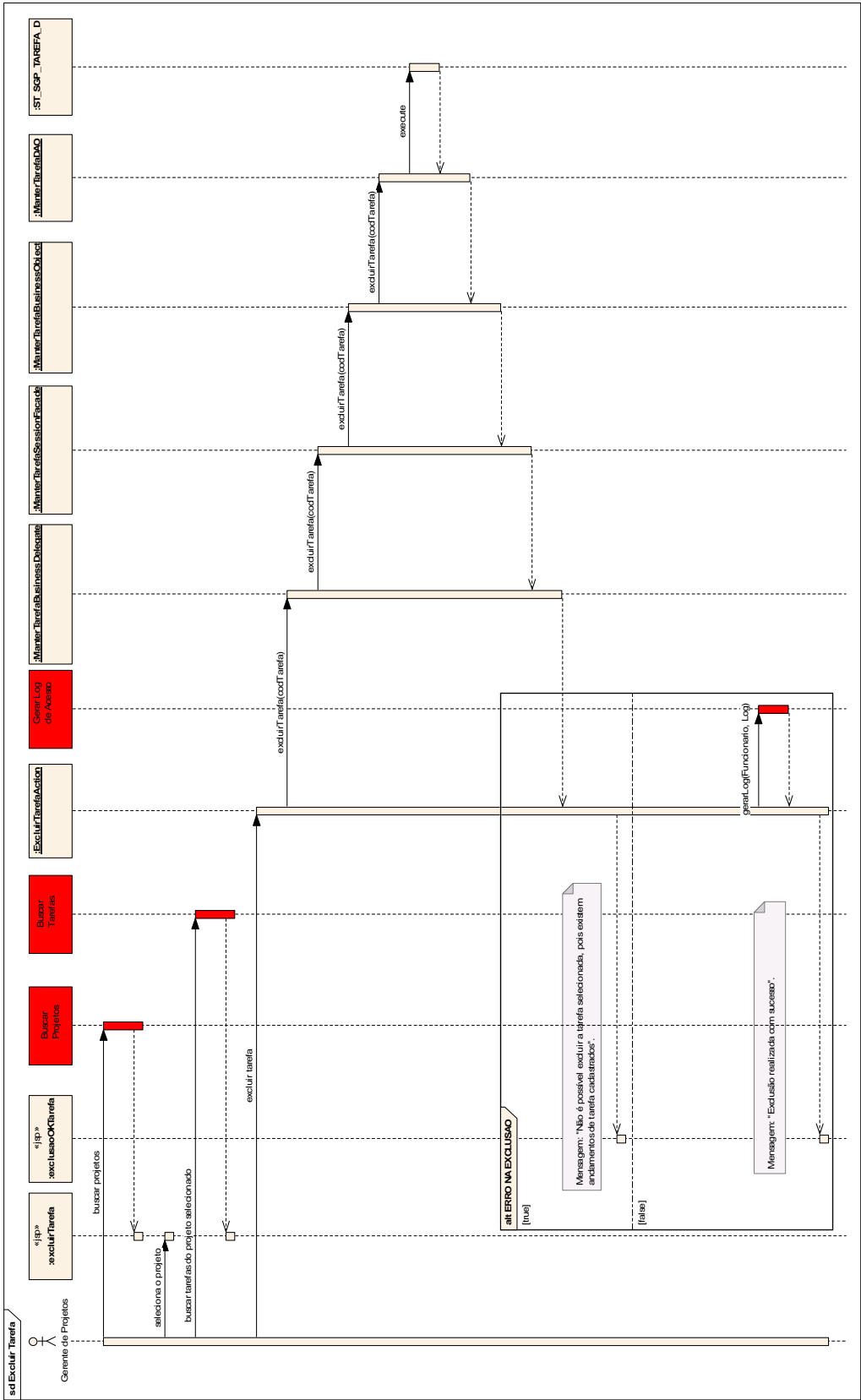
Buscar Funcionários

Alterar Funcionário

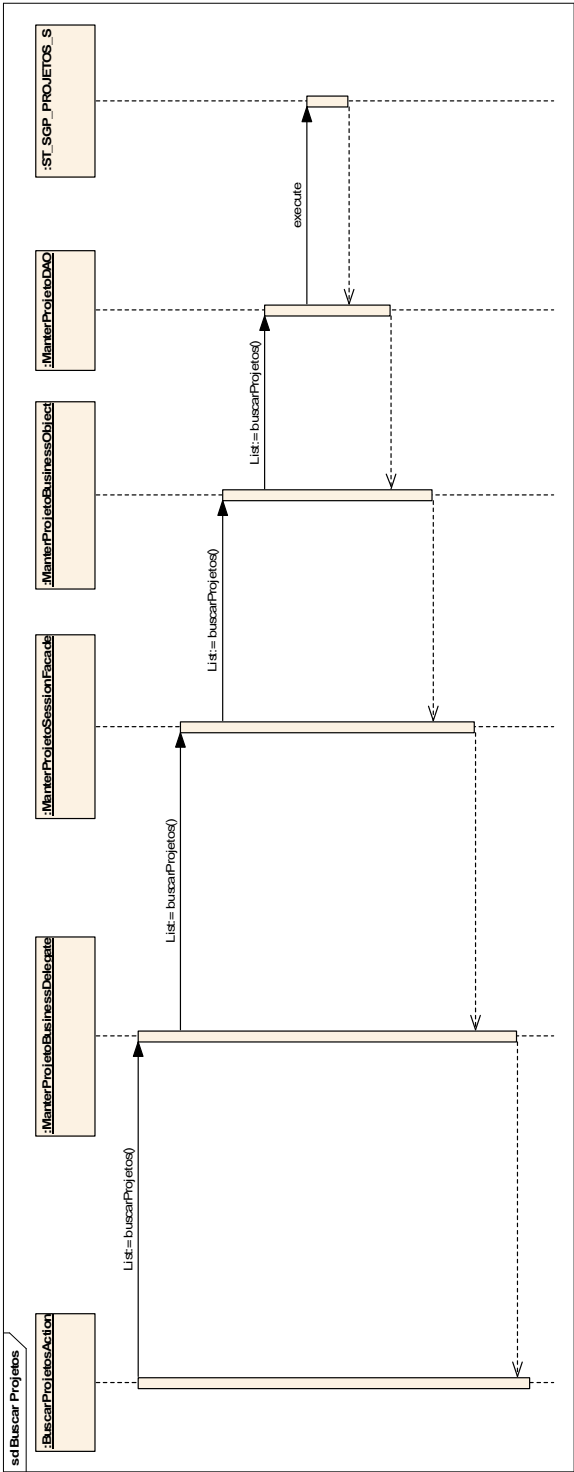
Cadastrar Funcionário

Desativar Funcionário

B2.1.2. UC02 – Vincular Permissões

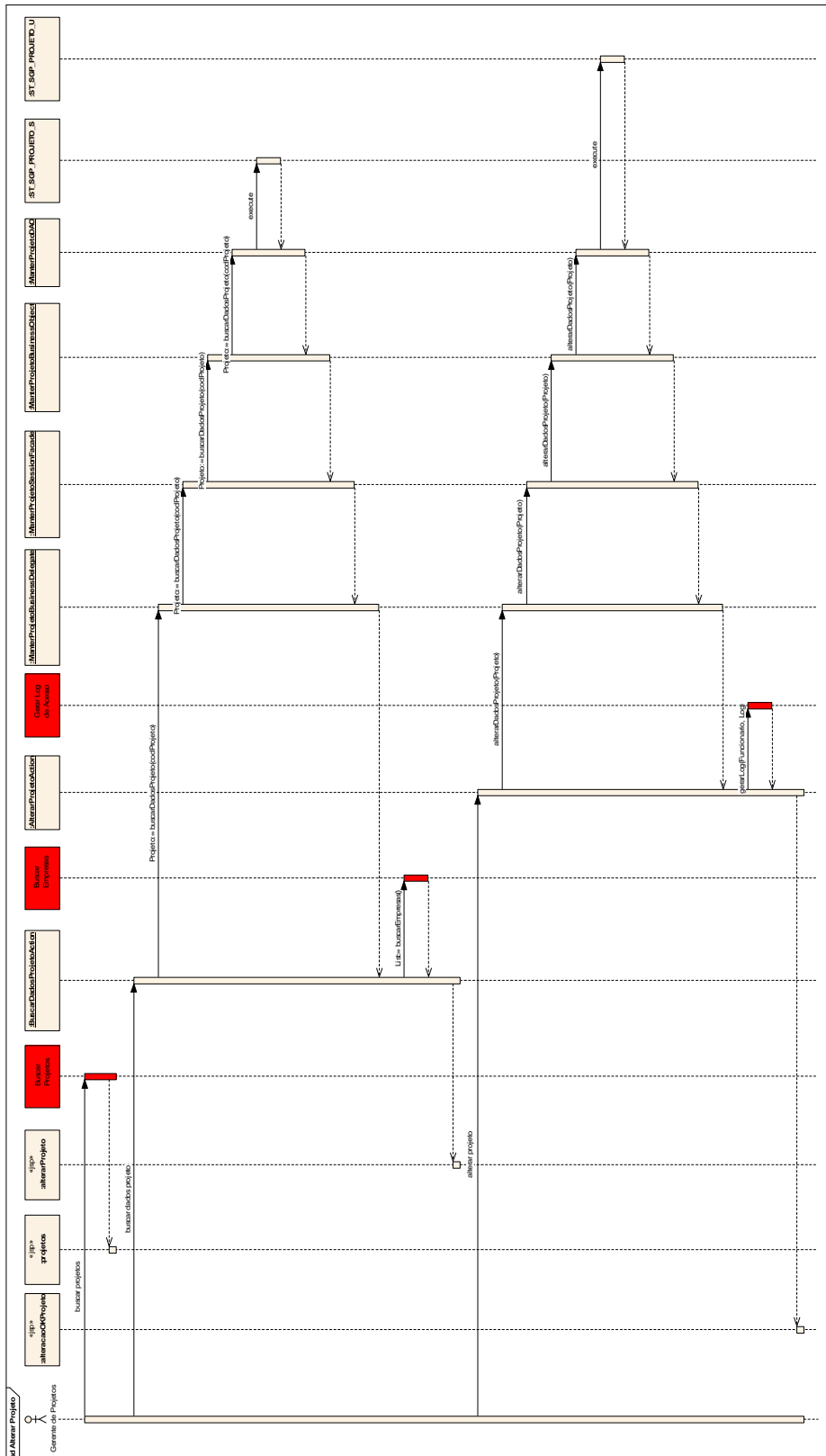
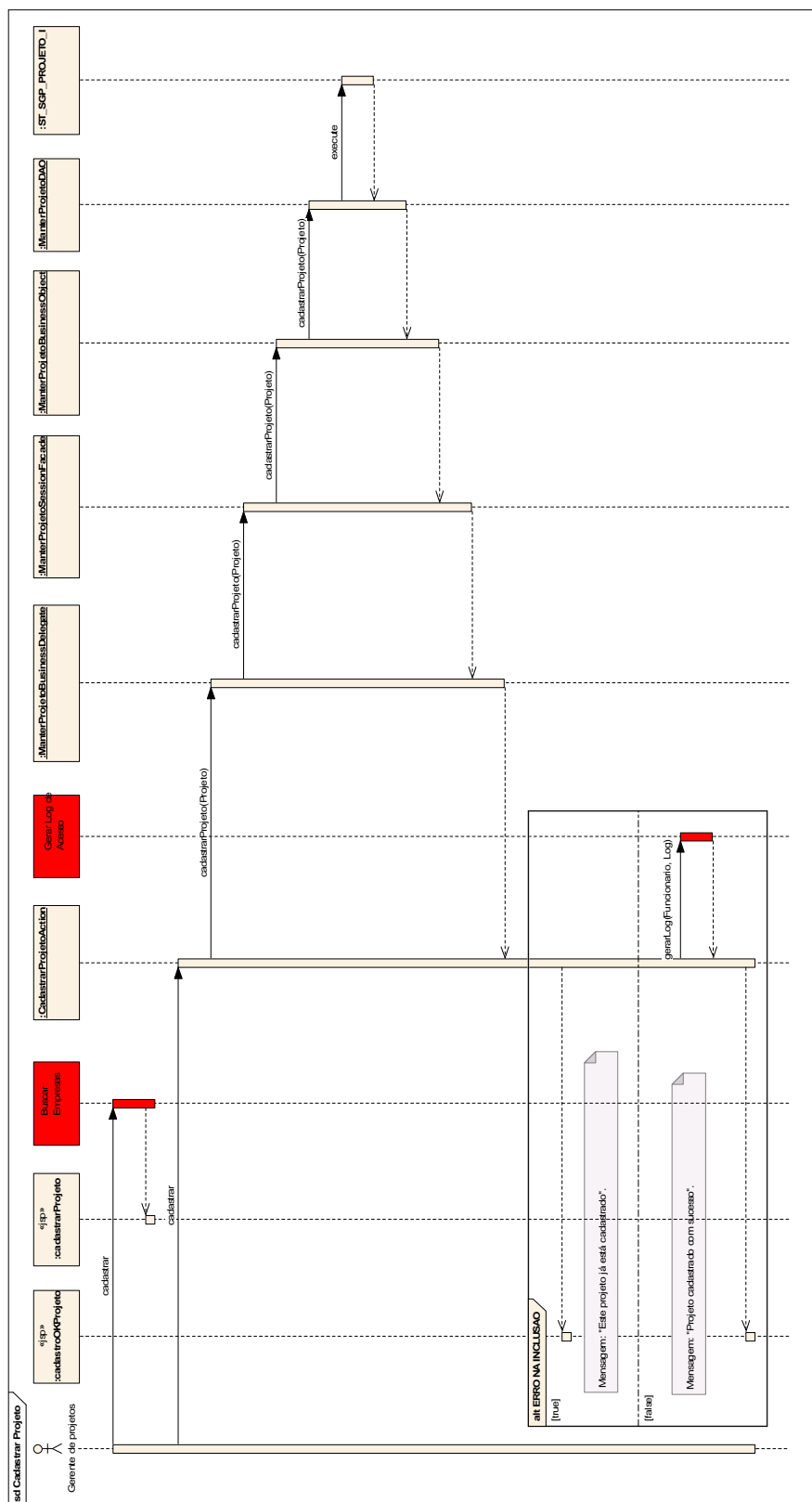
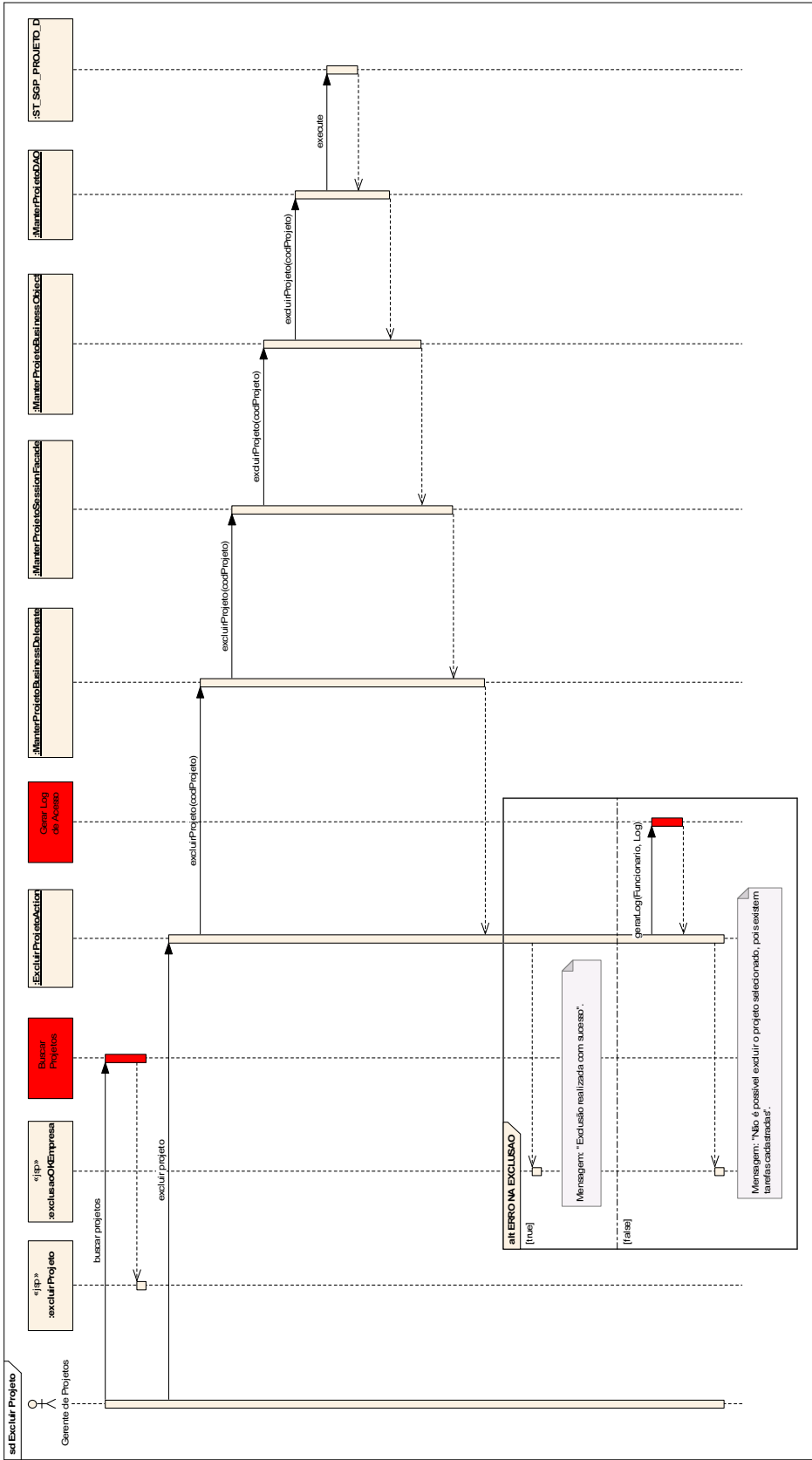
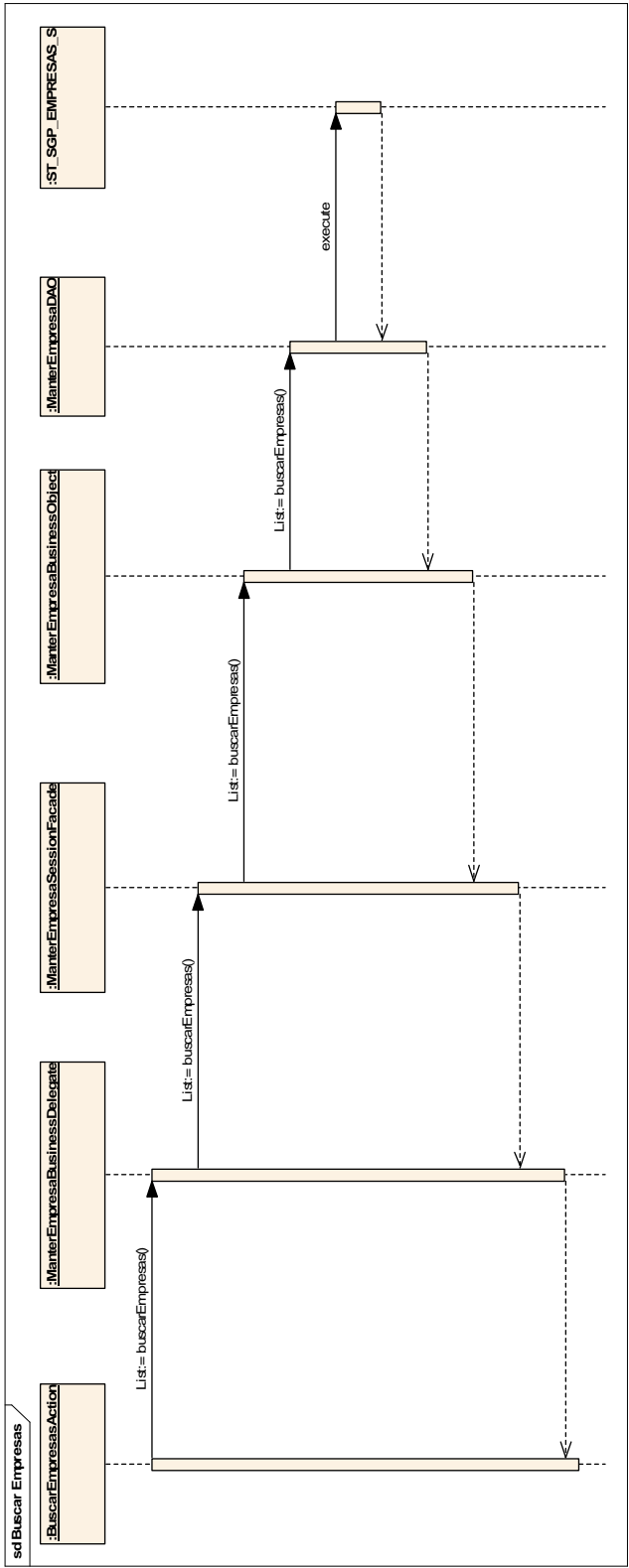
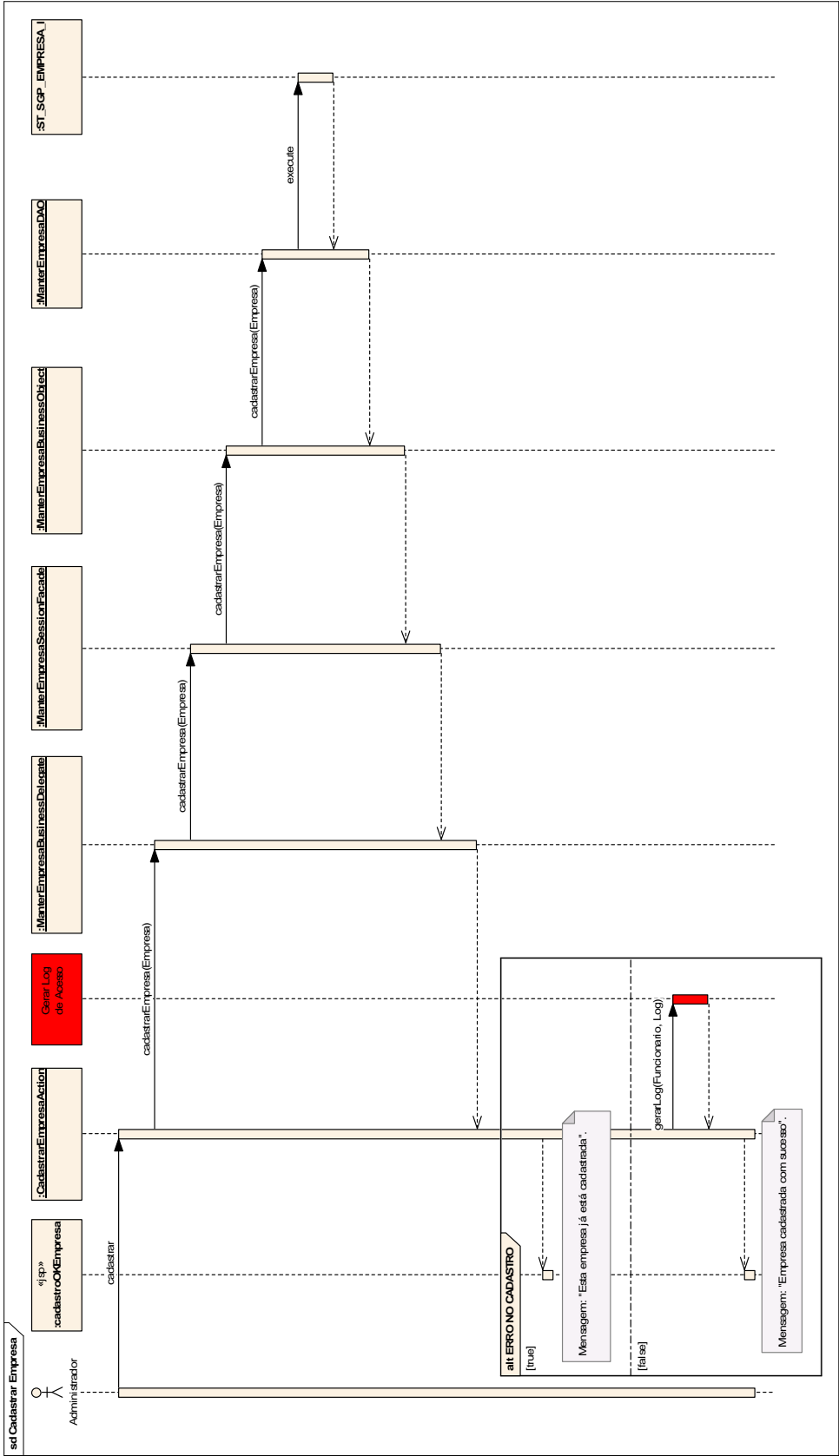
B2.1.3. UC03 – Vincular Funcionários

B2.1.4. UC04 - Manter Tarefas

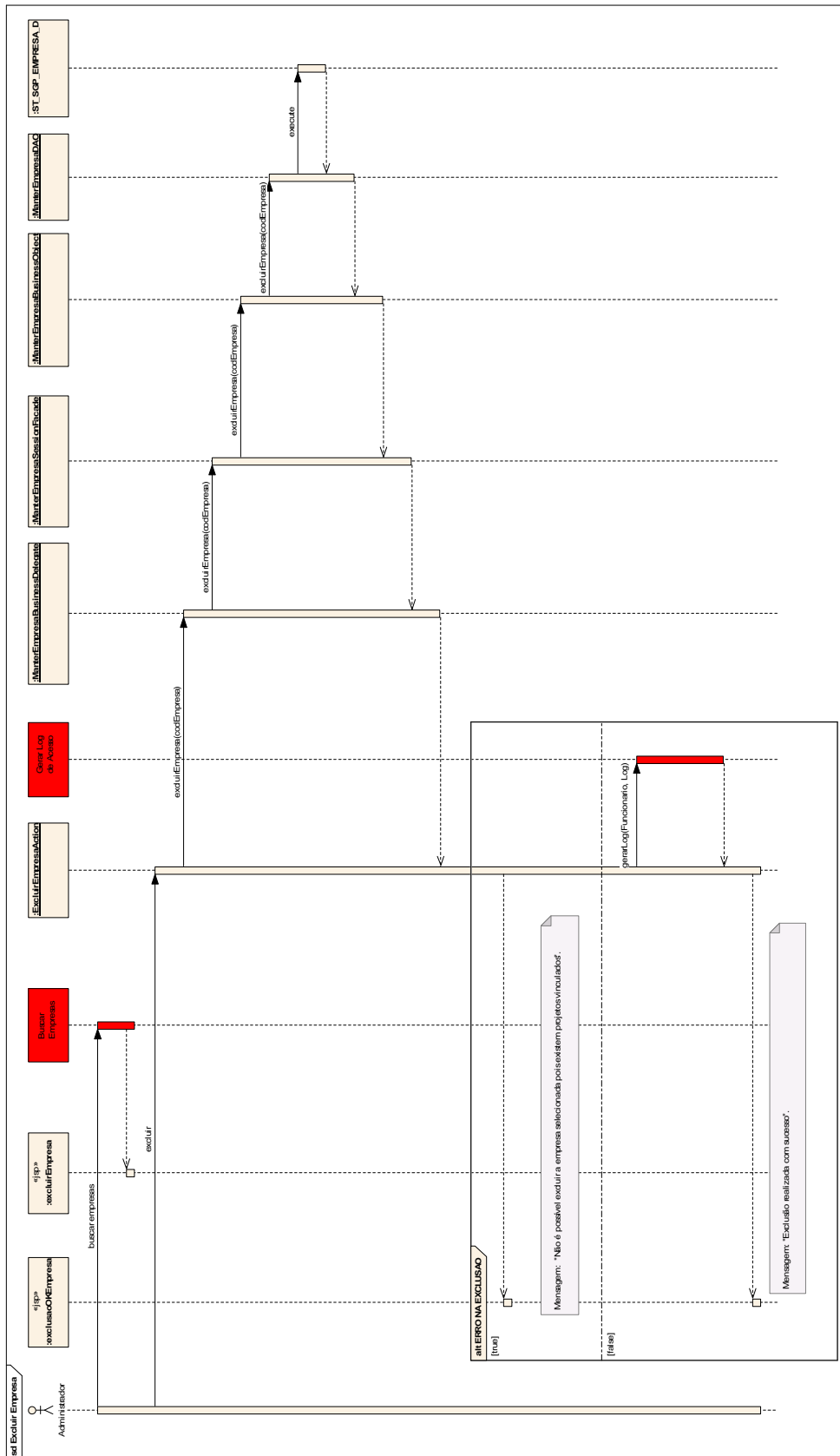
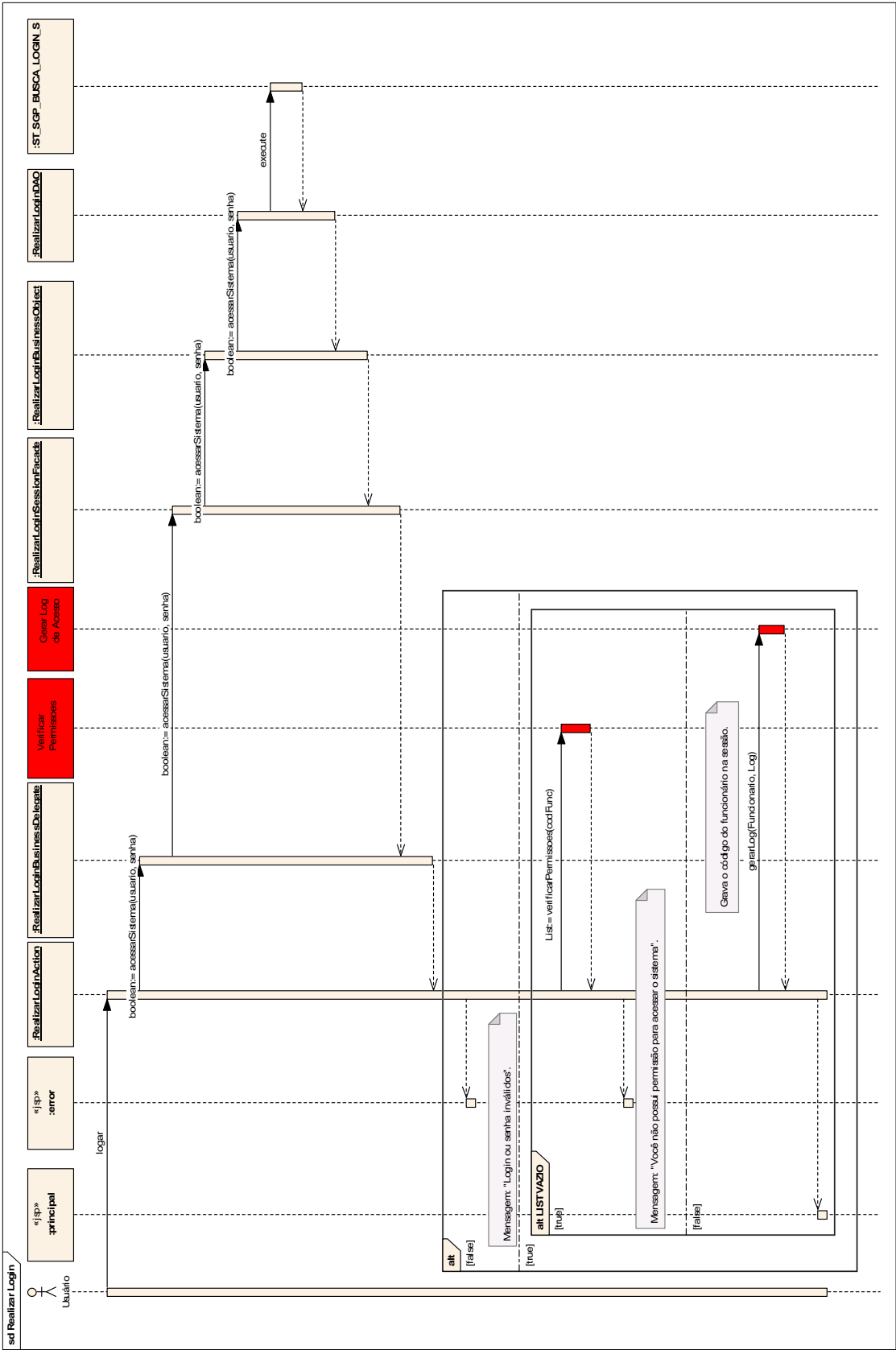
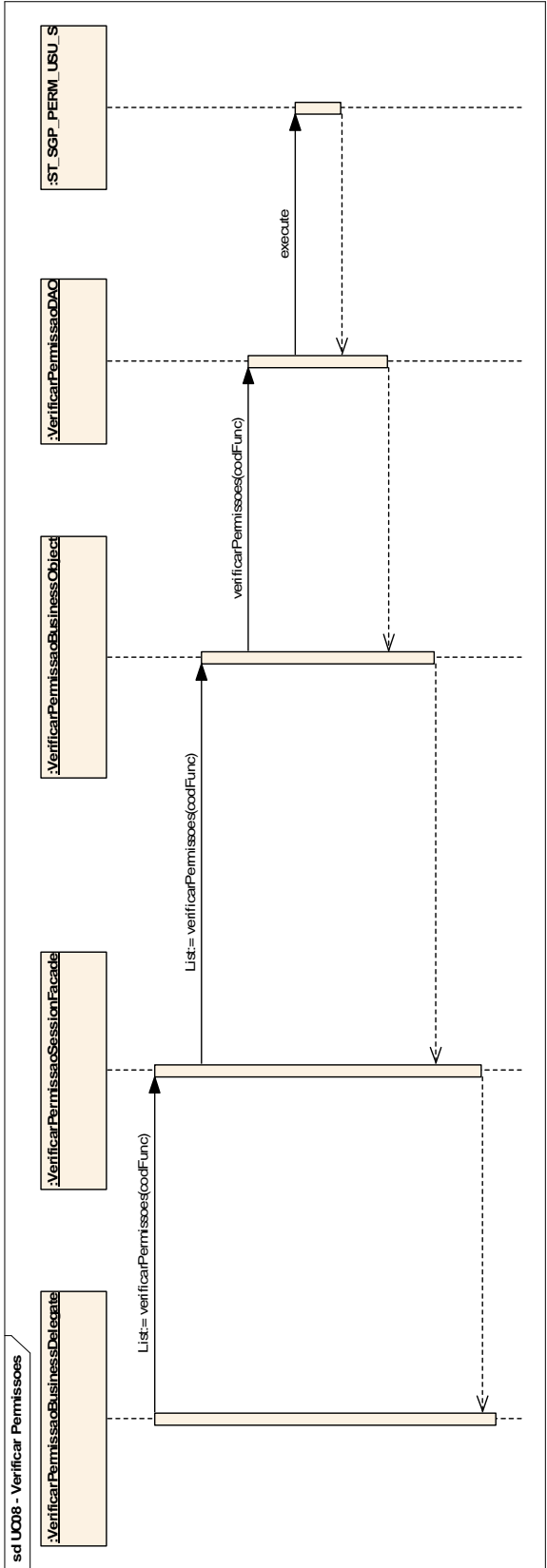
Buscar Tarefas

Alterar Tarefa

Cadastrar Tarefa

Excluir Tarefa

B2.1.5. UC05 – Manter Projetos

Buscar Projetos

Alterar Projeto

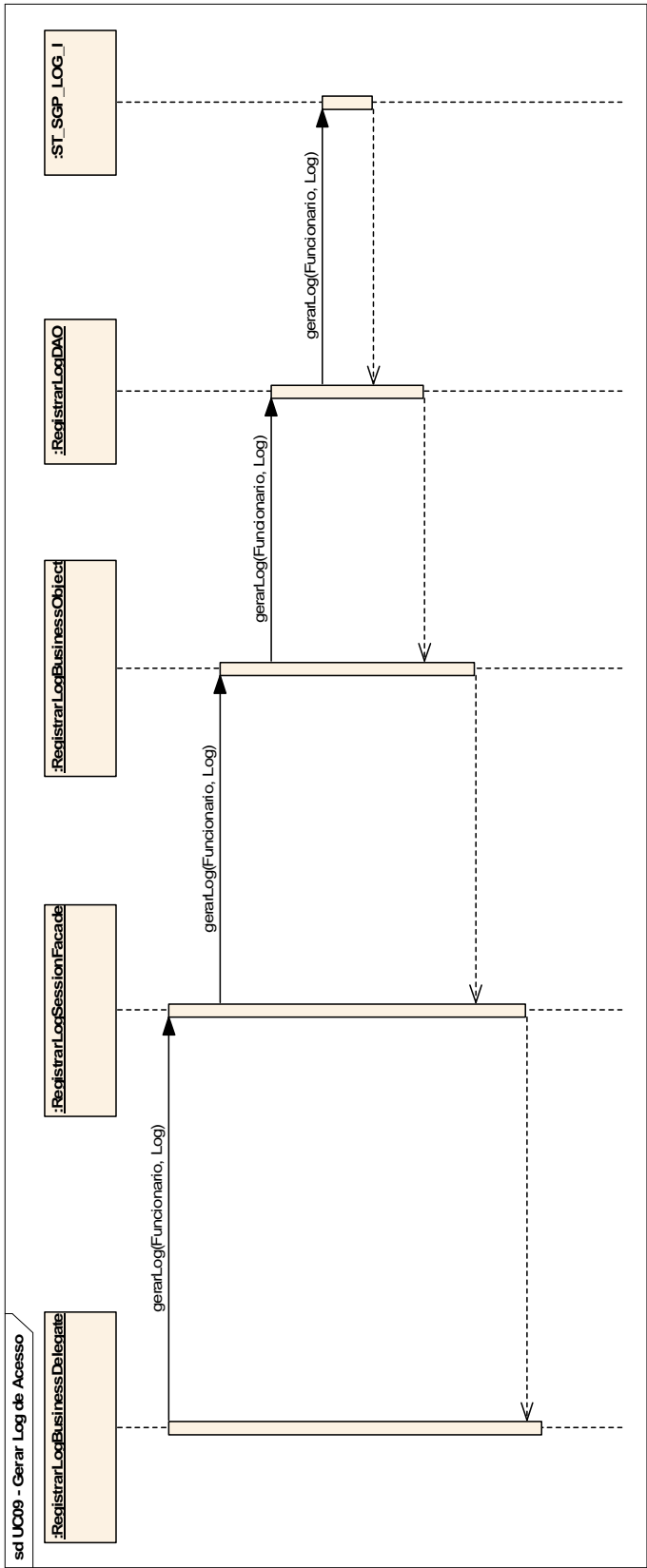
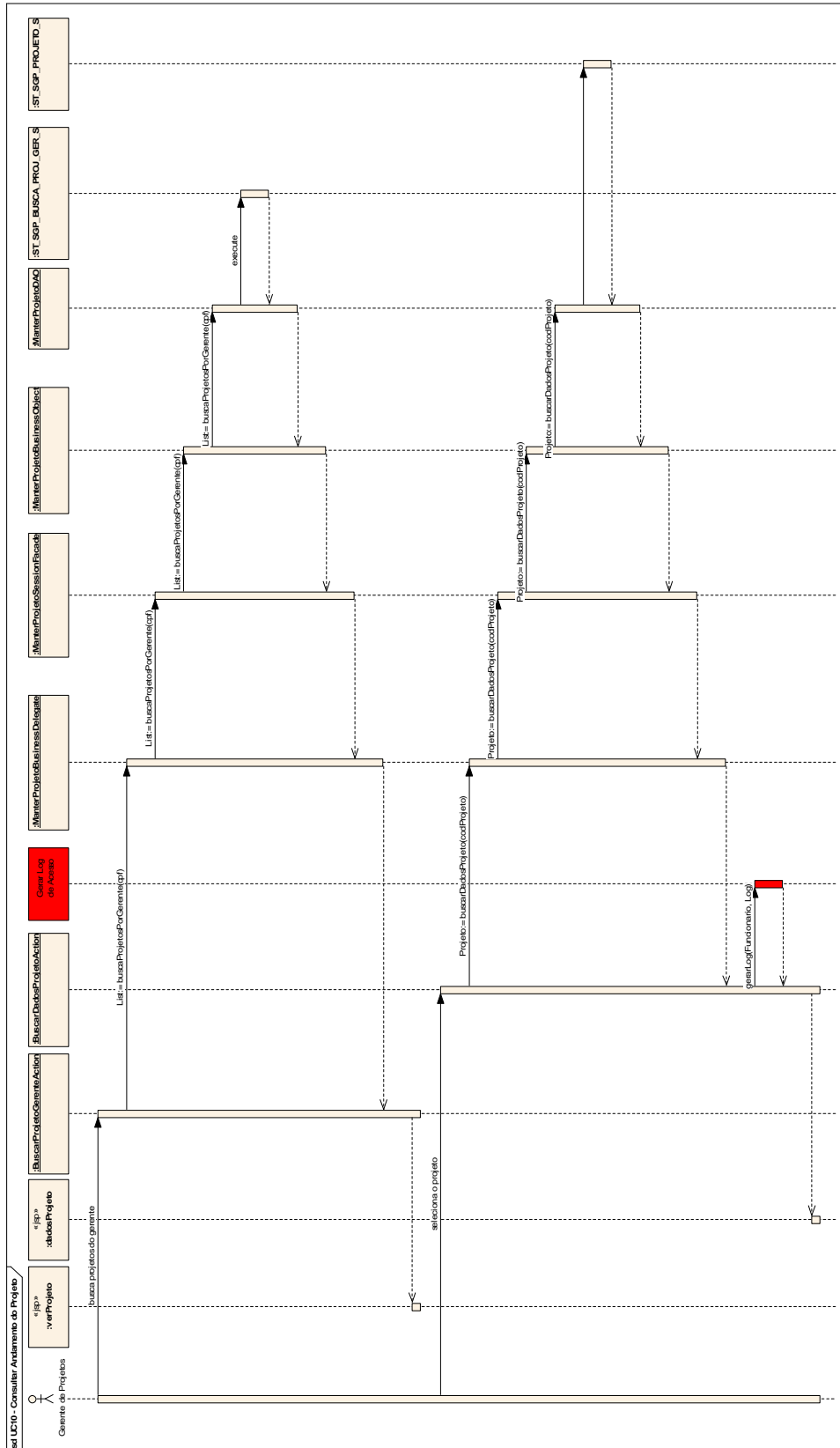
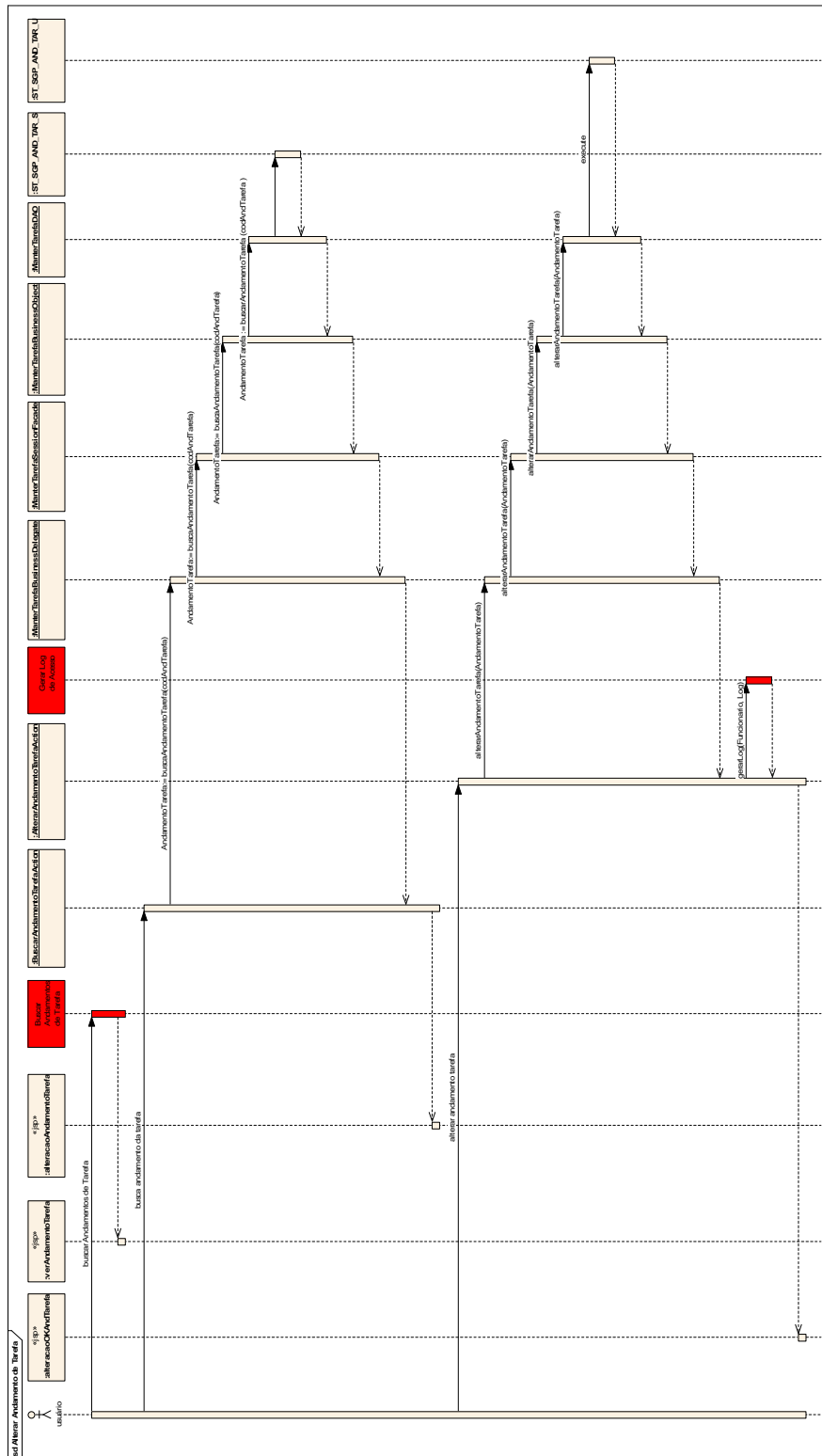
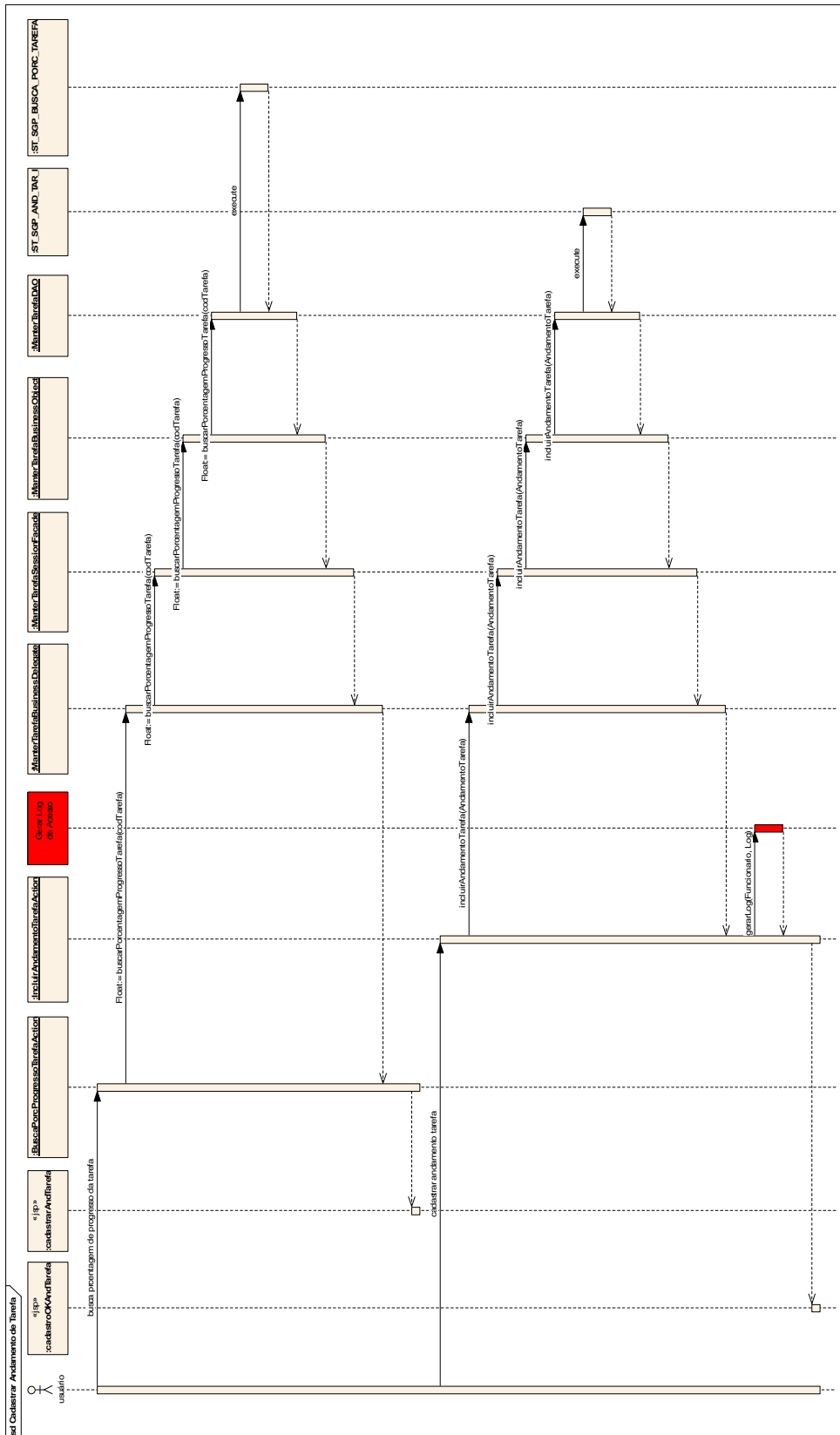
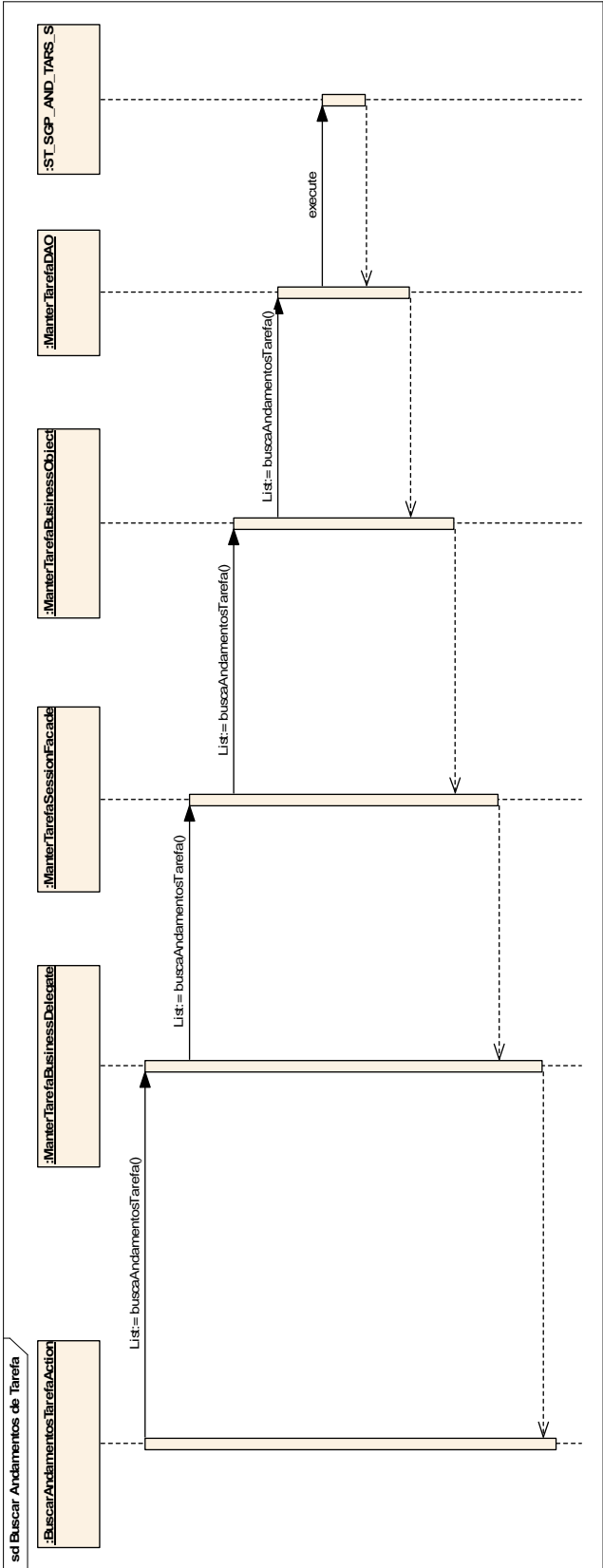
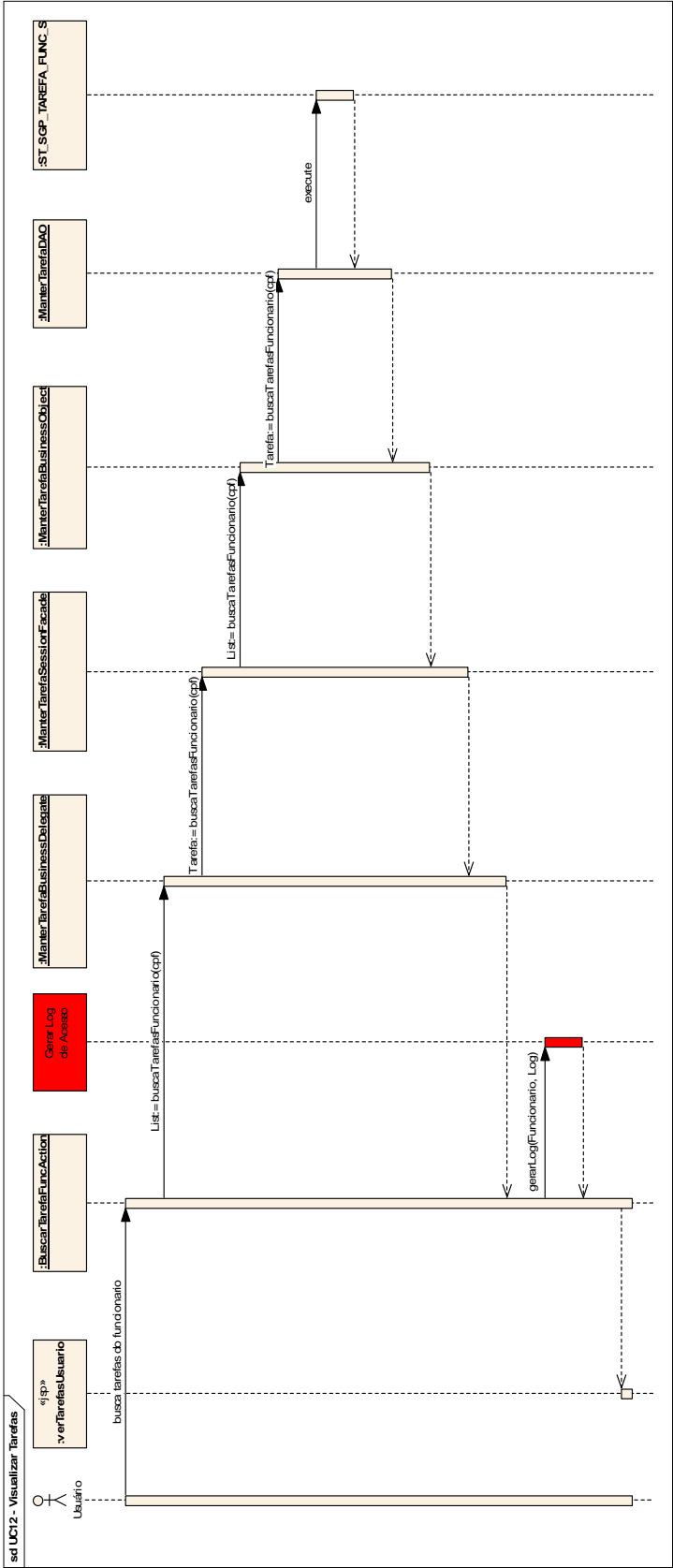
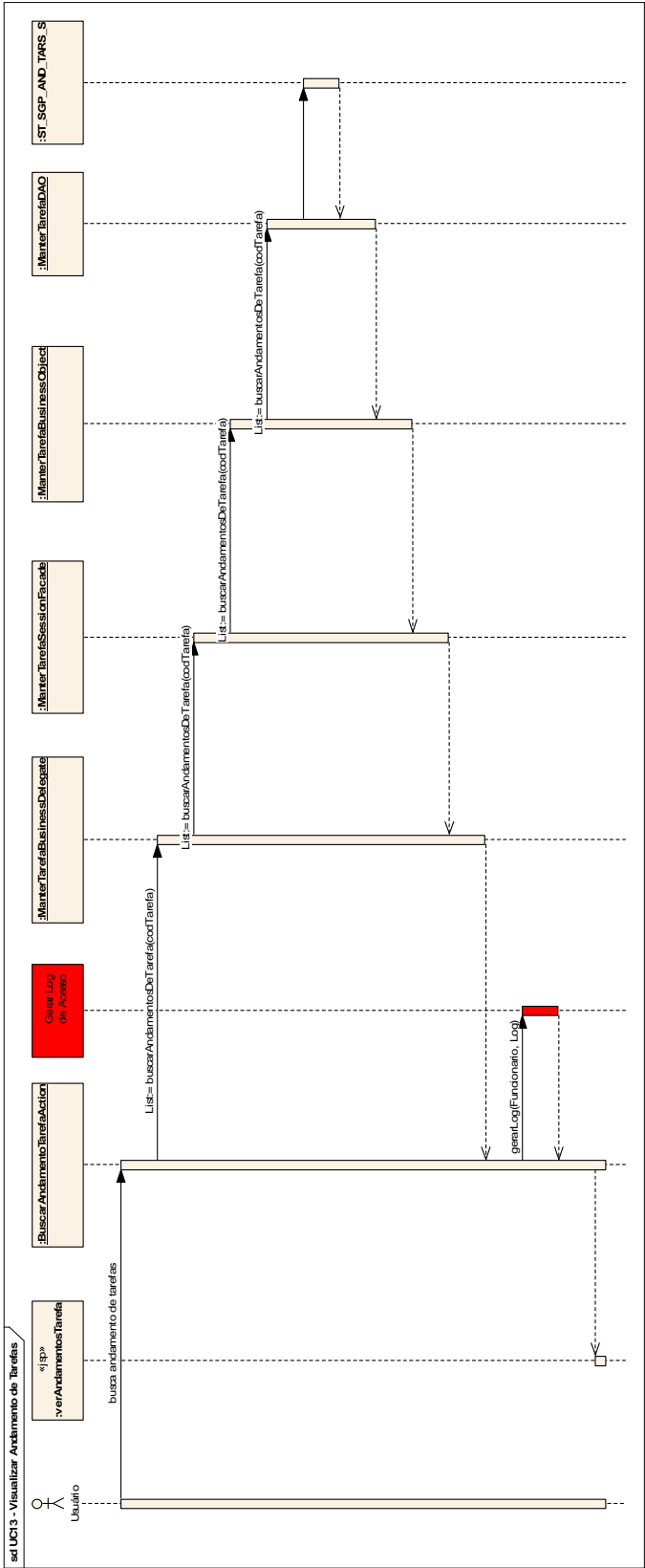
Cadastrar Projeto

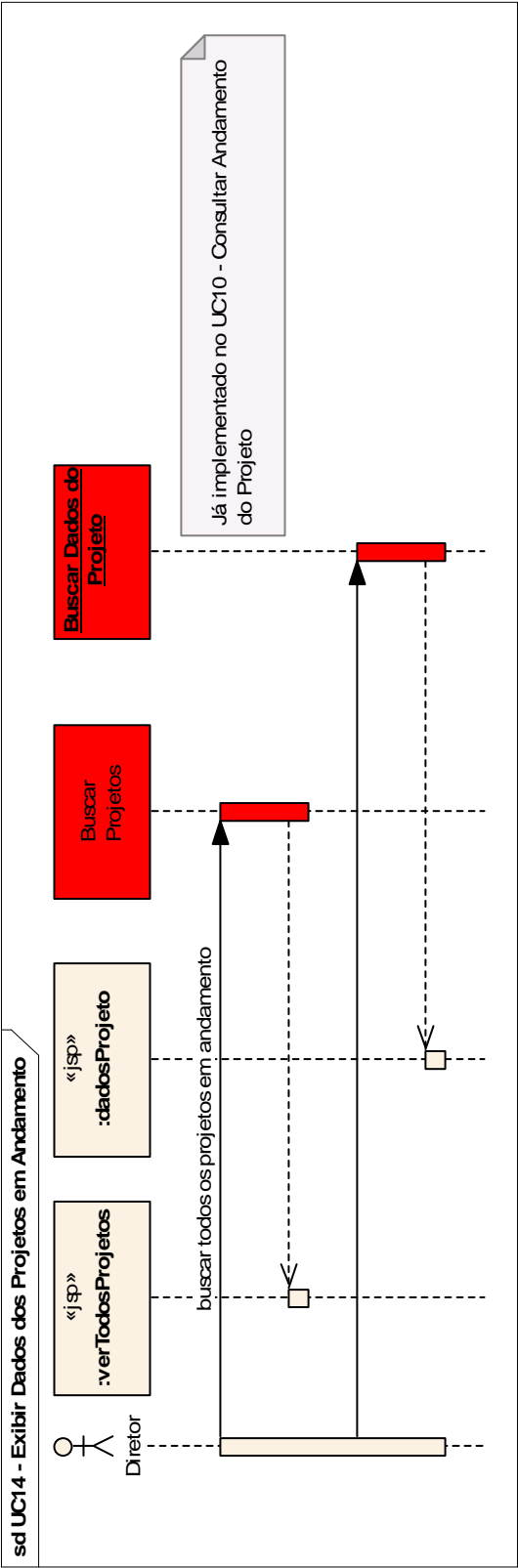
Excluir Projeto

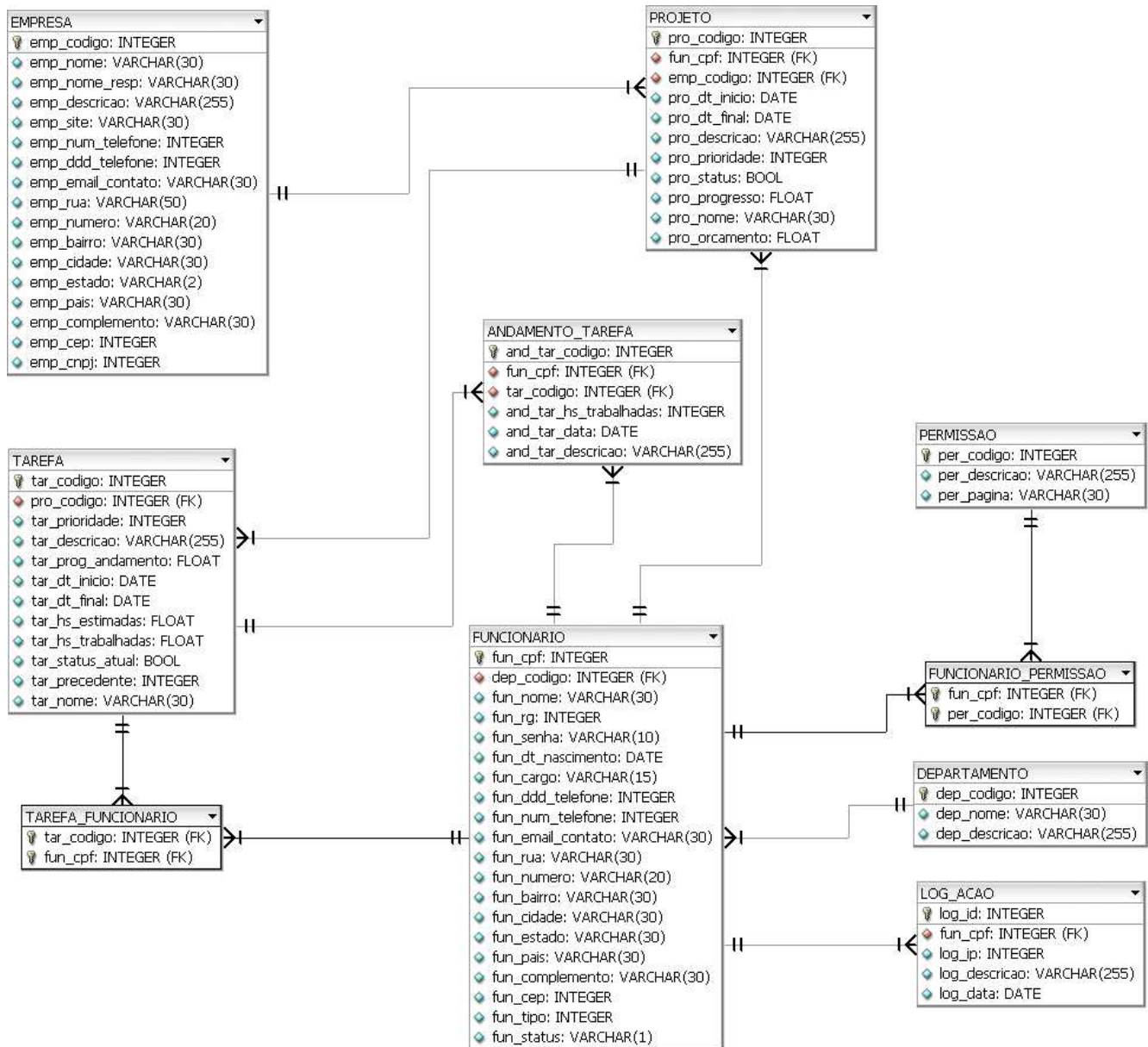
B2.1.6. UC06 – Manter Empresa

Buscar Empresas

Alterar Empresa

Cadastrar Empresa

Excluir Empresa

B2.1.7. UC07 – Realizar Login

B2.1.8. UC08 – Verificar Permissões

B2.1.9. UC09 – Gerar Log de Acesso

B2.1.10. UC10 – Consultar Andamento do Projeto

B2.1.11. UC11 – Manter Andamento da Tarefa

Alterar Andamento da Tarefa

Cadastrar Andamento da Tarefa

Buscar Andamento da Tarefa

B2.1.12. UC12 – Visualizar Tarefas

B2.1.13. UC13 – Visualizar Andamento de Tarefas

B2.1.14. UC14 – Exibir dados dos projetos em andamento

B2.2. Diagrama de classes

B3. Modelo de dados

B3.1. MER (Modelo Entidade Relacionamento)

B3.2. Dicionarização do Modelo

B3.2.1. Tabela: EMPRESA

Campo	Descrição
emp_codigo	Código da empresa
emp_nome	Nome da empresa
emp_nome_resp	Nome do responsável da empresa
emp_descricao	Breve descrição de como é a empresa
emp_site	Site da empresa
emp_num_telefone	Número de telephone
emp_ddd_telefone	DDD do número de telefone
emp_email_contato	E-mail de contato da empresa
emp_rua	Rua onde localiza a empresa
emp_numero	Número do local onde a empresa está situada
emp_bairro	Bairro onde a empresa está situada
emp_cidade	Cidade onde a empresa está situada
emp_estado	Estado onde a empresa está situada
emp_pais	País onde a empresa está situada
emp_complemento	Complemento do endereço da empresa
emp_cep	CEP da rua onde está localizada a empresa
emp_cnpj	CNPJ da Empresa

B3.2.2. Tabela: PROJETO

Campo	Descrição
pro_codigo	Código do projeto
emp_codigo	Código da empresa que está solicitando o projeto
pro_dt_inicio	Data de início do projeto
pro_dt_final	Data final de término do projeto
pro_descricao	Breve descrição do projeto, suas características, etc
pro_prioridade	Prioridade do projeto (3 – Alta; 2 – Média; 1 – Baixa)
pro_status	Status do projeto (A – Ativo; I – Inativo)
pro_progresso	Porcentagem de progresso de término do projeto
pro_nome	Nome do projeto
pro_orcamento	Valor orçado do projeto
fun_cpf	CPF do Gerente de Projetos

B3.2.3. Tabela: TAREFA

Campo	Descrição
tar_codigo	Código da tarefa.
pro_codigo	Código do projeto que a tarefa pertence
tar_prioridade	Prioridade da tarefa (3 – Alta; 2 – Média; 1 – Baixa)
tar_descricao	Breve descrição da tarefa, características, etc
tar_prog_andamento	Porcentagem de progresso de término da tarefa
tar_dt_inicio	Data de início da tarefa
tar_dt_final	Data de término da tarefa
tar_hs_estimadas	Quantidade de horas estimadas para executar a tarefa
tar_hs_trabalhadas	Quantidade de horas trabalhadas na tarefa
tar_status_atual	Status da tarefa (A – Ativo; I – Inativo)
tar_precedente	Tarefa que antecede esta.
tar_nome	Nome da tarefa

B3.2.4. Tabela: ANDAMENTO_TAREFA

Campo	Descrição
and_tar_codigo	Código do andamento da tarefa
tar_codigo	Código da tarefa que pertence o andamento da tarefa
and_tar_hs_trabalhadas	Quantidade de horas trabalhadas no andamento da tarefa
and_tar_data	Data do andamento da tarefa
and_tar_descricao	Descrição do andamento da tarefa
fun_cpf	Código do funcionário que efetuou o andamento da tarefa.

B3.2.5. Tabela: FUNCIONARIO

Campo	Descrição
fun_cpf	CPF e código do funcionário
fun_nome	Nome do funcionário
fun_rg	Registro Geral do funcionário
fun_senha	Senha de acesso ao sistema
fun_dt_nascimento	Data de nascimento do funcionário
fun_cargo	Cargo na empresa
fun_ddd_telefone	Código DDD do telefone
fun_num_telefone	Número do telefone
fun_email_contato	E-mail para contato
fun_rua	Endereço do funcionário
fun_numero	Número da casa do funcionário
fun_bairro	Bairro onde o funcionário mora
fun_cidade	Cidade onde o funcionário mora

fun_estado	Estado onde o funcionário mora
fun_pais	País onde o funcionário vive
fun_complemento	Complemento do endereço
fun_cep	CEP do endereço
dep_codigo	Código do departamento
fun_tipo	Tipo de funcionário (1 – Administrador, 2 – Diretor, 3 – Gerente de Projetos, 4 – Usuário)
fun_status	Status do Funcionários (A – Ativo, I – Inativo)

B3.2.6. Tabela: DEPARTAMENTO

Campo	Descrição
dep_codigo	Código do departamento
dep_nome	Nome do departamento
dep_descricao	Descrição do departamento

B3.2.7. Tabela: PERMISSAO

Campo	Descrição
per_codigo	Código da permissão
per_pagina	Página JSP do sistema
per_descricao	Descrição da permissão

B3.2.8. Tabela: LOG_ACAO

Campo	Descrição
log_id	Código do log
log_descricao	Descrição do log
log_ip	IP do computador de onde está gerando o log
log_data	Data da geração do log
fun_cpf	Código do funcionário

B4. Guia de programação

B4.1. Padrões de programação em Java

B4.1.1. Objetivo

Este documento descreve um conjunto de padrões, convenções e diretrizes para escrever um código Java sólido. Eles se baseiam em princípios de engenharia de melhorados. Além disso, com o uso desses padrões de codificação, sua produtividade como desenvolvedor Java deverá aumentar significativamente. A experiência demonstra que o tempo dedicado à escrita de códigos de alta qualidade desde o início facilitará a modificação do código durante o processo de desenvolvimento. Por fim, o uso de um conjunto comum de padrões de codificação garante maior consistência e aumenta a produtividade das equipes de desenvolvedores.

Primeira e Última Diretriz

Use o bom senso. Quando não houver nenhuma regra ou diretriz disponível, quando a regra não puder ser aplicada por motivos óbvios ou quando todo o resto falhar, use o bom senso e verifique os princípios fundamentais. Essa regra se sobrepõe a todas as outras. Ter bom senso é *obrigatório*.

B4.1.2. Padrões de Codificação

Os padrões de codificação para Java são importantes, porque garantem maior consistência de seu código e do código de seus companheiros de equipe. Maior consistência gera códigos mais fáceis de serem entendidos, desenvolvidos e mantidos. Dessa forma, você reduz o custo total dos aplicativos criados.

Lembre-se de que seu código Java continuará a existir por muito tempo ainda, mesmo depois de você estar envolvido em outros projetos. Uma meta importante durante o desenvolvimento é garantir que você possa passar seu trabalho para outro desenvolvedor ou outra equipe de desenvolvedores, a fim de que eles possam continuar a manter e melhorar seu trabalho sem precisarem investir um esforço desnecessário para entender o código que você criou. Códigos de difícil compreensão correm o risco de serem descartados e reescritos.

Convenções de nomeação

As convenções de nomeação serão abordadas junto com os padrões. Portanto, vamos definir alguns fundamentos:

- **Use descritores completos que descrevam com exatidão o atributo e a classe.** Por exemplo, use nomes como **primeiroName**, **totalProjetos** ou **qtdeClientes**. Embora nomes como **x1**, **y1** ou **fn** sejam fáceis de digitar por serem pequenos, eles não fornecem qualquer indicação do que representam, dificultando a compreensão, a manutenção e a melhoria do código.
- **Use a terminologia aplicável ao domínio em questão.** Se os usuários se referirem aos clientes como empresa, use o termo **Empresa** para a classe, e não **Cliente**. Muitos desenvolvedores cometem o erro de criar termos genéricos para conceitos quando já existem termos satisfatórios em uso no mercado ou no domínio.
- **Combine letras maiúsculas e minúsculas para facilitar a leitura dos nomes.** Use letras minúsculas em geral, mas maiúscula para a primeira letra dos nomes de classe e de interface, assim como para a primeira letra de palavras não-iniciais. [KAN97]
- **Não abuse das abreviações; use-as de forma inteligente.** Isso significa que você deve manter uma lista de formas abreviadas padrão (abreviações), escolhê-las com discernimento e usá-las consistentemente. Por exemplo, se você quiser usar uma forma abreviada da palavra "número", escolha **num**, documente sua preferência (não importa qual) e use somente essa.
- **Evite nomes longos (< 15 caracteres é o ideal).** Embora o nome da classe CadastroFuncionarioBusinessDelegate possa parecer satisfatório no momento em que é criado, ele é muito longo por seguir uma padronização. Classes que representam os Bean's, devem seguir esta convenção.
- **Evite nomes semelhantes ou cuja única diferença sejam as letras maiúsculas ou minúsculas.** Por exemplo, os nomes de variável persistentObject e persistentObjects não devem ser usados ao mesmo tempo, nem anSqlDatabase e anSQLDatabase.

- **Evite sublinhados iniciais ou finais.** Nomes com sublinhados iniciais ou finais geralmente são reservados para uso do sistema e não devem ser utilizados para nomes criados pelo usuário, exceto se o pré-processador assim definir. Mais importante ainda é que os sublinhados confundem e dificultam a digitação. Portanto, tente evitá-los sempre que possível.

Convenções de documentação

Abordaremos também as convenções de documentação. Para isso, vamos definir alguns fundamentos:

- **Comentários contribuem para a clareza do código.** O motivo de documentar o código é torná-lo mais compreensível para você, seus colaboradores e outros desenvolvedores que venham a substituí-lo.
- **Se seu programa não merece ser documentado, provavelmente não merece ser executado.** [NAG95]
- **Evite enfeites, ou seja, não use comentários como faixas.** Você deve escrever códigos legíveis, e não códigos enfeitados. Além disso, como muitas fontes usadas para exibir e imprimir o código são proporcionais e outras não, não é possível alinhar as caixas corretamente.
- **Mantenha a simplicidade dos comentários.** Os melhores comentários são anotações simples e diretas. Você não tem de escrever um livro, precisa apenas fornecer informações suficientes para que os usuários possam entender o código.
- **Crie a documentação antes de escrever o código.** A melhor maneira de criar a documentação é redigir os comentários antes de escrever o código. Dessa forma, você poderá pensar sobre como o código funcionará antes de escrevê-lo e garantirá o preparo da documentação. Como opção, você também pode documentar o código à medida que o escreve. Como a documentação facilita a compreensão do código, você poderá utilizá-la ao desenvolver o código. Se você pretende investir tempo no preparo da documentação, deve pelo menos tirar algum proveito disso. [AMB98]
- **Documente não apenas o que está sendo feito, mas também por que está sendo feito.** Por exemplo, o código do Exemplo 1 abaixo mostra que um desconto de 5% foi oferecido aos pedidos iguais ou superiores a R$ 1.000,00. Por que fazer isso? Existe alguma regra de negócio que imponha desconto aos pedidos maiores? Esse desconto é por tempo limitado ou é um programa permanente? O programador original estava apenas sendo generoso? Você só saberá responder se o motivo estiver documentado, seja no próprio código-fonte ou em um documento externo.

<u>Tipos de comentários em JAVA</u>

Java tem três estilos de comentários:

1. Comentários de documentação, que começam com /** e terminam com */

2. Comentários de estilo C, que começam com /* e terminam com */
3. Comentários em uma única linha, que começam com // e seguem até o fim da linha do código-fonte

A tabela a seguir é um sumário de uso sugerido para cada tipo de comentário, assim como também o são vários exemplos.

Comentário	**Uso**	**Exemplo**
Documentação	Use os comentários de documentação imediatamente antes de declarações de interfaces, classes, funções de membro e campos para documentá-los. Esses comentários serão processados pelo *javadoc*. Veja a seguir como criar documentação externa para uma classe.	/** Cliente: é qualquer pessoa ou organização a quem vendemos serviços e produtos. @author S.W. Ambler */
Estilo C	Use os comentários de estilo C para documentar linhas de código que não são mais aplicáveis, mas que você ainda quer manter caso os usuários mudem de idéia ou porque você talvez queira desativá-los temporariamente durante um processo de depuração.	/* Esse código foi desativado por B. Gustafsson no dia 4 de junho de 1999, porque foi substituído pelo código precedente. Exclua-o após dois anos caso ainda não seja aplicável. . . . (o código-fonte) */

Linha única	Use comentários em uma única linha internamente nas funções de membro para documentar a lógica do negócio, as seções do código e as declarações de variáveis temporárias.	// Aplicar desconto de 5% a todas as
		// faturas acima de R$ 1.000,00 em função de
		// campanha geral de descontos iniciada em
		// fevereiro de 1995.

É importante que sua organização defina um padrão para mostrar como os comentários de estilo C e os de uma única linha serão usados e siga-o de forma consistente. Use um tipo para

documentar a lógica do negócio e outro para documentar o código antigo. Use os comentários de uma única linha para a lógica do negócio, pois você pode incluir a documentação na mesma linha do código (esse procedimento é conhecido como inserção em linha). Use os comentários de estilo C para documentar o código antigo e desnecessário, pois eles permitem desativar várias linhas de código de uma só vez. Como os comentários de estilo C são bastante semelhantes aos comentários de documentação, para evitar confusão, não os use em outros locais.

Uma visão geral do javadoc

No Sun Java Development Kit (JDK) está incluído um programa denominado *javadoc*, que processa arquivos de código Java e produz documentação externa para os programas Java na forma de arquivos HTML. O programa *javadoc* suporta um número limitado de marcas, palavras reservadas que marcam o início de uma seção da documentação. Consulte a documentação do *javadoc* JDK para obter mais detalhes.

Marca	Usado para	Finalidade
@author name	Classes, Interfaces	Indica os autores de determinado trecho do código. Use uma marca por autor.
@deprecated	Classes, Funções de Membro	Indica que a API da classe ficou obsoleta e, portanto, não deve mais ser usada.
@exception name description	Funções de Membro	Descreve as exceções que uma função de membro aciona. Use uma marca por exceção e informe o nome completo da classe referente à exceção.
@param name description	Funções de Membro	Usada para descrever um parâmetro passado para uma função de membro, inclusive seu tipo ou classe e sua utilização. Use uma marca por parâmetro.
@return description	Funções de Membro	Descreve o valor de retorno (se houver) de uma função de membro. Indique o tipo ou a classe e os possíveis usos do valor de retorno.
@since	Classes, Funções de Membro	Indica o tempo de existência do item, ou seja, desde o JDK 1.1.
@see ClassName	Classes, Interfaces, Funções de Membro, Campos	Gera um link de hipertexto na documentação para a classe especificada. Você pode - e provavelmente deve - usar um nome de classe

		totalmente qualificado.
@see ClassName#member functionName	Classes, Interfaces, Funções de Membro, Campos	Gera um link de hipertexto na documentação para a função de membro especificada. Você pode - e provavelmente deve - usar um nome de classe totalmente qualificado.
@version text	Classes, Interfaces	Indica as informações sobre versão de determinado trecho do código.

A maneira como você documenta o código causa grande impacto tanto na sua produtividade quanto na produtividade de todos que posteriormente serão responsáveis pela manutenção e melhoria do código. Com a documentação do código no início do processo de desenvolvimento, sua produtividade será maior porque você será forçado a pensar sobre a lógica empregada antes de aplicá-la ao código. Além disso, quando você retorna ao código que escreveu dias ou semanas antes, pode determinar facilmente o que tinha em mente ao escrevê-lo, pois os procedimentos já estão documentados.

Nunca se esqueça de que o código escrito hoje poderá ser usado por muitos anos e provavelmente será mantido e melhorado por outra pessoa. Você deve procurar criar um código legível e compreensível, porque esses fatores facilitam a manutenção e a implementação de melhorias.

B4.1.3. Padrões para Métodos

Nunca se esqueça de que o código escrito hoje poderá ser usado por muitos anos e provavelmente será mantido e melhorado por outra pessoa. Você deve procurar criar um código legível e compreensível, porque esses fatores facilitam a manutenção e a implementação de melhorias.

Nomeação de métodos

As funções de membro devem ser nomeadas através de uma descrição completa, que combine letras maiúsculas e minúsculas, sendo que a primeira letra de toda palavra não-inicial deve ser maiúscula. Também é comum usar um verbo forte e ativo para a primeira palavra de uma função de membro.

Exemplos:

verificarProjetos( );

buscarAndamentoTarefas( );

cadastrarEmpresas(Empresa empresa);

Esta convenção resulta métodos, cuja finalidade pode ser freqüentemente determinada por seus nomes. Embora envolva um pouco mais de digitação por parte do desenvolvedor, porque geralmente os nomes são longos, essa convenção é conveniente por aumentar a compreensão do código.

Nomeação de Métodos de Acesso

Abordaremos mais detalhadamente os acessos, isto é, métodos que obtêm e definem os valores de campos (campos ou propriedades). As convenções de nomeação para os acessos, entretanto, estão resumidas a seguir.

Getters

Getters são métodos que retornam o valor de um campo. A palavra "get" deve ser usada como prefixo do nome do campo, a menos que o campo seja boolean. Nesse caso, o prefixo do nome será "is", e não "get".

Exemplos:

getNomeProjeto( );

getCpfFuncionario( );

getEnderecoEmpresa( );

De acordo com essa convenção de nomeação, fica óbvio que uma função de membro retornará um campo de um objeto. No caso de boolean getters, fica óbvio que o valor retornado será verdadeiro ou falso. [DES97]

Setters

Os setters são métodos que modificam os valores de um campo. Use a palavra "set" como prefixo para o nome do campo, seja qual for o tipo de campo.

Exemplos:

setNomeProjeto(String nomeProjeto);

setCpfFuncionario(Integer cpf);

setEnderecoEmpresa(Endereco enderecoEmpresa);

Com essa convenção de nomeação, fica óbvio que um método definirá o valor de um campo de um objeto. [DES97]

Nomeação de Construtores

Os construtores são métodos que executam a inicialização necessária quando um objeto é criado pela primeira vez. Os construtores sempre recebem o mesmo nome da classe a que pertencem. Por exemplo, um construtor da classe **Tarefa** será **Tarefa( )**. O uso de letras maiúsculas e minúsculas é o mesmo.

Essa convenção de nomeação é definida pela Sun Microsystems e deve ser seguida rigorosamente.

Visibilidade de métodos

Para garantir um bom design no qual você diminua o acoplamento entre classes, adote como regra prática geral ser o mais restritivo possível ao definir a visibilidade de um método. Se um método não precisa ser público, defina-a como protegido. Se não precisar ser protegido, defina-a como privado.

Visibilidade	Descrição	Uso adequado
pública	Um método público pode ser disparado por outro método em outro objeto ou classe.	Quando o método tem de ser acessado por objetos e classes fora da hierarquia de classes na qual o método está definido.

protegida	Um método protegido pode ser disparado por qualquer método da classe na qual ela está definida ou por qualquer subclasse dessa classe.	Quando p método fornece comportamento que é necessário internamente à hierarquia de classes, mas não externamente.
privada	Um método privado só pode ser disparado por outros métodos da classe na qual ela está definida, mas não nas subclasses.	Quando o método oferece comportamento que é específico para a classe. Os métodos privados geralmente são o resultado da refatoração (ou reorganização) do comportamento de outros métodos contidas na classe para encapsular um comportamento específico.

Documentação de métodos

A maneira como você documenta um método costuma ser um fator decisivo para determinar se ele será entendido e, portanto, se poderá ser mantida e ampliada.

<u>O cabeçalho dos métodos</u>

Todo método em Java deve incluir algum tipo de cabeçalho (denominado documentação da função de membro) na parte superior do código-fonte que documente todas as informações que são vitais para sua compreensão. Essas informações incluem, entre outras, as seguintes questões:

- **O que o método executa e por quê.** Ao documentar o que um método executa, você permite que outros usuários determinem se o código poderá ser reutilizado. Documentar os motivos pelos quais ela executa permite que outros usuários insiram seu código no contexto. Você permite também que outros determinem se uma nova mudança deverá ser realmente efetuada em um trecho do código (talvez o motivo para a nova mudança entre em conflito com o motivo pelo qual o código foi escrito inicialmente).
- **Qual método deve ser passado como parâmetro.** Também é preciso indicar quais parâmetros (se houver) devem ser passados para um método e como eles serão usados.

Essas informações são necessárias para que os outros programadores saibam quais informações devem ser passados para um método.

- **O que é retornado por um método.** Documente o que o método retorna (caso ela ofereça algum tipo de retorno) para que outros programadores possam usar o valor ou o objeto de retorno corretamente.
- **Erros conhecidos.** Qualquer problema pendente com um método deve ser documentado para que os outros desenvolvedores conheçam os pontos fracos e as dificuldades do método. Se um erro se aplicar a mais de um método de uma classe, ele deverá ser documentado para a classe, e não para o método.
- **Documente toda e qualquer exceção que seja acionada por um método para que outros programadores saibam o que o código precisará obter.**
- Decisões de visibilidade. Se você achar que sua opção de visibilidade para um método será questionada pelos outros desenvolvedores (talvez você tenha criado um método como público, mesmo que nenhum outro objeto a dispare por enquanto), documente sua decisão. Isso tornará seu raciocínio claro para os outros desenvolvedores, evitando que eles percam tempo preocupando-se por que você fez algo questionável.
- **Como um método altera o objeto.** Indique se um método altera um objeto. Por exemplo, o método **withdraw()** de uma conta bancária modifica o saldo dessa conta. Essas informações são necessárias para que outros programadores Java saibam exatamente como o disparo de um método afetará o objeto de destino.
- **Evite o uso de cabeçalhos que contenham informações** como autor, números de telefone, datas de criação/modificação e localização de unidades (ou nome de arquivo), porque elas ficam rapidamente obsoletas. Coloque os avisos de direitos autorais no fim da unidade. Por exemplo, os leitores não querem percorrer duas ou três páginas de texto que não são úteis para a sua compreensão do programa, nem querem passar por texto que não contenha qualquer informação específica do programa, como um aviso de direitos autorais. Evite o uso de barras verticais ou quadros e caixas fechados, pois eles apenas criam confusão visual e são difíceis de manter a consistência. Use uma ferramenta de gerenciamento de configuração para manter um histórico de unidades.
- **Exemplos de como disparar o método, se apropriado.** Uma das maneiras mais fáceis de determinar como um trecho do código funciona é verificar um exemplo. Considere a possibilidade de incluir um ou dois exemplos sobre como disparar um método.
- **Precondições e pós-condições aplicáveis**. Precondição é uma restrição segundo a qual um método funcionará corretamente. Pós-condição é uma propriedade ou afirmativa que será verdadeira depois de concluída a execução de um método. [MEY88] As precondições e pós-condições descrevem de várias formas as suposições feitas durante a criação de um método e [AMB98] definem com precisão os limites de uso desses.
- **Todas as questões de simultaneidade.** Simultaneidade é um conceito novo e complexo para muitos desenvolvedores e, na melhor das hipóteses, é um tópico antigo e complexo para programadores com experiência em simultaneidade. Em suma, se você usar as características de simultaneidade da programação Java, deverá documentá-las integralmente. [LEA97] Sugere que, quando uma classe inclui métodos sincronizados e não sincronizados, você deve documentar o contexto de execução em que se baseia o

método, principalmente se ele requer acesso irrestrito para que os outros desenvolvedores possam utilizá-la com segurança. Se um setter (um método que atualiza um campo) de uma classe que implementa a interface **Runnable** não estiver sincronizado, você deverá documentar os motivos pelos quais não está. Por fim, caso você sobreponha ou sobrecarregue um método e altere sua sincronização, também deve documentar o motivo.

- **Você só deve documentar aquilo que contribua para a clareza do código**. Não é preciso documentar todos os fatores descritos acima para toda e qualquer método, porque nem todos os fatores são aplicáveis a todas os métodos. Entretanto, vários deles devem ser documentados para cada método criado.

Documentação interna

Além da documentação de métodos, você também precisa incluir comentários que descrevam o seu trabalho. O objetivo é facilitar a compreensão, a manutenção e a implementação de melhorias nas funções de membro.

Há dois tipos de comentários que podem ser usados para documentar as especificações internas do código: os comentários de estilo C (/* e */) e os comentários em uma única linha (//). Como abordado anteriormente, você deve considerar a possibilidade de escolher um estilo de comentários para documentar a lógica do negócio do código e outro para comentar os códigos não necessários. Utilize comentários em uma única linha para a lógica do negócio, pois esse estilo pode ser usado para linhas de comentários inteiras e para comentários in-line que continuem até o fim de uma linha de código. Use os comentários de estilo C para documentar linhas de código desnecessário, pois eles facilitam a retirada de várias linhas do código com apenas um comentário. Além disso, como os comentários de estilo C são muito semelhantes aos comentários de documentação, sua utilização pode causar confusão, reduzindo a compreensão do código. Sendo assim, não os utilize com freqüência.

Internamente, você sempre deve documentar o seguinte:

- **Estruturas de controle.** Descreva cada estrutura de controle, como sentenças de comparação e ciclos. Você não deve precisar ler o código inteiro em uma estrutura de controle para determinar o que ele executa. Basta verificar uma ou duas linhas de comentários que o antecedem.
- **O que o código executa e por quê.** Ao verificar um trecho do código, você sempre pode saber o que ele faz. Contudo, se o código não for óbvio, raramente será possível definir o motivo de determinada ação. Por exemplo, você pode verificar uma linha de código e determinar facilmente que um desconto de 5% está sendo aplicado ao valor total de um pedido. Isso é fácil. O que não é fácil é perceber POR QUE esse desconto está sendo aplicado. Obviamente, há uma regra de negócio que determina a aplicação do desconto. Portanto, essa regra deve ser pelo menos mencionada no código, para que os outros desenvolvedores possam entender o motivo.
- **Variáveis locais.** Embora esse assunto seja abordado mais detalhadamente no capítulo 5, cada variável local definida em uma função de membro deve ser declarada em sua

própria linha de código e, geralmente, deve ter um comentário in-line que descreva seu uso.

- **Código difícil ou complexo.** Se você perceber que não é possível reescrevê-lo ou caso não tenha tempo para isso, documente completamente os códigos complexos de uma função de membro. Como regra prática geral, se o código não for óbvio, é preciso documentá-lo.
- **A ordem de processamento.** Se houver instruções no código que precisem ser executadas em uma ordem definida, garanta que sejam documentadas. [AMB98] Não há nada pior do que efetuar uma simples modificação em um trecho do código para depois perceber que ele não funciona mais. As horas investidas na procura do problema significam que você só conseguiu desordenar tudo.
- **Documente as chaves de fechamento.** Com freqüência, você verá que há estruturas de controle contidas em outras estruturas de controle que, por sua vez, também estão contidas em estruturas de controle. Embora você deva evitar escrever códigos desse tipo, pode haver situações em que seja preciso fazê-lo. O problema é que fica confuso determinar a qual chave de finalização o caractere **}** pertence, a qual estrutura de controle. A boa notícia é que alguns editores de código suportam um recurso que, quando você seleciona uma chave de abertura, a chave de fechamento é automaticamente destacada. A má notícia é que nem todos os editores suportam esse recurso. De acordo com a nossa experiência, marcar as chaves de finalização com um comentário in-line, como **//end if**, **//end for**, **//end switch**, facilita bastante a compreensão do código.

Técnicas para escrever códigos legíveis

Esta seção aborda várias técnicas que ajudam a separar os desenvolvedores profissionais de outros tipos de codificadores. As técnicas são:

- **Documentar o código.** Lembre-se de que, se não valer a pena documentar o código, não valerá a pena mantê-lo. [NAG95] Se você aplicar corretamente os padrões e as diretrizes de documentação propostos neste documento, obterá códigos de melhor qualidade.
- **Criar parágrafos ou recuos no código.** Uma maneira de melhorar a legibilidade de uma função de membro é criar parágrafos ou recuos no código dentro do escopo de um bloco de código. Qualquer código contido entre chaves - os caracteres { e } - compõe um bloco. A idéia básica é que o código contido em um bloco seja uma unidade uniformemente recuada. A convenção Java sugere que a chave de abertura seja colocada na linha seguinte ao proprietário do bloco e que a chave de fechamento seja recuada em um nível. O importante, segundo [LAF97], é que a sua organização escolha um estilo de recuo e se atenha a ele. Use o mesmo estilo de recuo que o ambiente de desenvolvimento Java usa para o código que ele gera.
- **Usar espaço em branco.** Algumas linhas vazias (denominadas espaço em branco) adicionadas ao código Java podem aumentar a sua legibilidade, dividindo-o em pequenas seções de fácil assimilação. [VIS96] Sugere o uso de uma única linha vazia para separar grupos lógicos do código (como estruturas de controle), com duas linhas vazi-

as para separar as definições de função de membro. Sem o espaço em branco, a leitura e a compreensão ficam mais difíceis.

- **Seguir a regra dos 30 segundos.** Os outros programadores devem conseguir entender o que a função de membro executa, bem como o motivo e a maneira dessa execução em menos de 30 segundos. Se isso não for possível, é sinal de que o código é muito difícil de manter e deve ser melhorado. Trinta segundos, nada mais. A prática demonstra que, se uma função de membro ocupa mais de uma tela, provavelmente ela é muito grande.
- **Criar linhas de comandos simples e curtas.** O código deve ter uma execução por linha. Na época das placas de inserção, fazia sentido tentar o máximo de funcionalidade em uma única linha de código. Sempre que você tentar mais de uma execução em uma única linha de código, estará dificultando o entendimento. Por que fazer isso? Queremos facilitar a compreensão do código para também facilitar sua manutenção e melhoria. Assim como uma função de membro deve desempenhar apenas uma ação, você também deve incluir apenas uma ação em uma única linha de código. Além disso, você deve escrever códigos que possam ser visualizados na tela [VIS96]. Você não deve precisar rolar a janela de edição para a direita a fim de ler toda a linha de código, inclusive do código que usa comentários in-line.
- **Especificar a ordem das operações.** Uma maneira realmente fácil de aumentar a legibilidade do código é usar parênteses para especificar a ordem exata das operações no código Java [NAG95] e [AMB98]. Se você precisa saber a ordem das operações de uma linguagem para entender o código-fonte, é sinal de que há algo muito errado. Isso é um problema principalmente nas comparações lógicas, em que AND e OR e diversos comparadores são usados em conjunto. Se você usar linhas de comandos simples e curtas, como sugerido anteriormente, isso não será mais um problema.

B4.1.4. Padrões para campos e propriedades

O termo *campo* usado aqui se refere a um campo que o Bean Development Kit chama de propriedade [DES97]. Campos são dados que descrevem um objeto ou uma classe. Os campos podem ser tipos de dados básicos (como seqüências de caracteres ou flutuantes) ou podem ser um objeto (como um cliente ou uma conta bancária).

Nomeação de campos

Use um descritor completo para nomear os campos [GOS96] e [AMB98], explicando o que cada campo representa. Campos que são conjuntos (como matrizes ou vetores) devem receber nomes no plural para indicar que representam vários valores.

Nomeação de constantes

Em Java, as constantes — valores que não mudam — são geralmente implementadas como campos de classes *finais estáticos*. A convenção reconhecida é o uso de palavras completas, todas em maiúsculas, separadas por sublinhados [GOS96].

Exemplos:

TIPO_FUNCIONARIO

COD_DEPTO

MAX_VALUE

A principal vantagem dessa convenção é ajudar a fazer a diferenciação entre constantes e variáveis.

Visibilidade de atributos

Quando os atributos são declarados como *protegidos*, existe a possibilidade de eles serem acessados diretamente por métodos das subclasses, aumentando de forma eficaz o acoplamento em uma hierarquia de classes. Como esse procedimento dificulta a manutenção e a implementação de melhorias nas classes, deve ser evitado. Os atributos nunca devem ser acessados diretamente. Para acessá-los, use métodos de acesso (veja abaixo).

Visibilidade	Descrição	Uso adequado
Público	Pode ser acessado por outro método em outro objeto ou classe.	Não crie atributos públicos.
Protegido	Pode ser acessado por método da classe	Não crie atributos protegidos.

	na qual ele está declarado ou por qualquer método definido em subclasses dessa classe.	
Privado	Pode ser acessado por métodos da classe na qual ele está definido, mas não nas subclasses.	Todos os atributos devem ser privados e acessados por métodos getter e setter (acessos).

Não ocultar nomes

Ocultamento de nome se refere à prática de atribuir a uma variável local, a um argumento ou a um atributo um nome igual (ou semelhante) ao de outro com escopo mais amplo. Por exemplo, se um atributo foi denominado **firstName**, não crie um parâmetro ou variável local denominado **firstName**, ou parecido com **firstNames** ou **fistName**. Isso dificultará a compreensão do código e ele ficará sujeito a erros, porque os outros desenvolvedores, ou até mesmo você, não lerão o código corretamente enquanto ele estiver sendo modificado, tornando mais difícil a detecção de erros.

Uso de métodos de acesso

Além das convenções de nomeação, a manutenibilidade dos campos é obtida pelo uso adequado de *métodos de acesso*, ou seja, métodos que oferecem a funcionalidade necessária para atualizar um campo ou acessar seu valor. Os métodos de acesso podem ser de dois tipos: *setters* (também conhecidos como *mutantes*) e *getters*. O setter modifica o valor de uma variável, enquanto o getter a obtém.

Nomeação de métodos de acesso

Os nomes dos métodos getter devem incluir "get" + nome do campo. Se o campo representar um valor boolean (verdadeiro ou falso), o nome do getter poderá ser "is" + nome do campo. Os nomes dos métodos setter devem ser "set" + nome do campo, seja qual for o tipo de campo. O nome de campo é sempre uma combinação de letras maiúsculas e minúsculas, com letra maiúscula no início de todas as palavras.

Exemplo:

Campo	**Tipo**	**Nome do getter**	**Nome do setter**
nomeProjeto	String	getNomeProjeto( )	setNomeProjeto( )

tarPrecedente	Integer	getTarPrecedente( )	setTarPrecedente( )
cpf	Integer	getCpf( )	setCpf( )
endereco	Objeto	getEndeteco( )	setEndeteco( )
permissoes	Collection	getPermissoes( )	setPermissoes( )

Visibilidade de métodos de acesso

Sempre que uma "classe externa" precisar de acesso a um campo é que o respectivo getter ou setter deverá ser público para que esta classe possa modificar ou obter determinado valor deste campo.

B4.1.5. Padrões para variáveis locais

Uma variável local é um objeto ou item de dados definidos no escopo de um bloco, geralmente uma função de membro. O escopo de uma variável local é o bloco no qual ela está definida. O padrão importante de codificação para as variáveis locais enfatiza a declaração e convenção de documentação, que tem como convenção os elementos abaixo:

- **Declarar uma variável local por linha de código.** Isso está consistente com uma instrução por linha de código e possibilita documentar cada variável com um comentário.
- **Documentar variáveis locais com um comentário in-line.** O uso de comentários in-line é um estilo no qual um comentário em uma única linha, denotado por //, vem logo após um comando na mesma linha de código (procedimento este denominado comentário de fim de linha). Documente para que, onde e por que uma variável local deve ser usada. Assim será mais fácil entender o código.
- **Usar variáveis locais para um único fim**. Sempre que você usa uma variável local por mais de um motivo, você efetivamente diminui a coesão e dificulta a compreensão. Isso também aumenta a probabilidade de introduzir erros no código causados por efeitos colaterais inesperados de valores anteriores de uma variável local no início do código. Com certeza, a reutilização de variáveis locais é mais eficiente, pois é necessário alocar menos memória. No entanto, a reutilização de variáveis locais diminui a manutenibilidade do código e o fragiliza. Geralmente, não vale a pena arriscar apenas por não precisar alocar mais memória.

B4.1.6. Padrões para classes, Interface e Pacotes

Uma classe é um template a partir do qual objetos são instanciados (criados). As classes contêm a declaração de campos (atributos) e métodos. As interfaces são a definição de uma assinatura comum, incluindo métodos e campos, que uma classe que implementa uma interface deve suportar. Um pacote é um conjunto de classes relacionadas.

Classes

A convenção Java padrão usa um descritor lingüístico completo, com a primeira letra em maiúscula e uma combinação de maiúsculas e minúsculas no resto do nome. Os nomes também precisam estar no singular.

As seguintes informações devem ser exibidas nos comentários de documentação imediatamente antes da definição de uma classe:

- **A finalidade da classe.** Os desenvolvedores precisam saber qual é o objetivo geral de uma classe para que possam determinar se ela atende às suas necessidades.
- **Erros conhecidos**. Se houver problemas pendentes com uma classe, eles devem ser documentados para que outros desenvolvedores entendam os pontos fracos e as dificuldades relacionados a essa classe. Além disso, o motivo para não corrigir o erro também precisa ser documentado.
- **O histórico de desenvolvimento ou manutenção da classe.** É bastante comum a inclusão de uma tabela de histórico com datas, autores e sumários das mudanças efetuadas em uma classe. Dessa forma, os programadores de manutenção têm idéia das modificações sofridas por uma classe, além de saber quem fez o quê na classe.

Interface

A convenção Java é nomear as interfaces com letras maiúsculas e minúsculas combinadas, sendo que a primeira letra de cada palavra deve ser maiúscula. A convenção Java mais aconselhável para o nome de uma interface é usar um adjetivo descritivo (como **Runnable** ou **Cloneable**), embora substantivos descritivos (como **Singleton** ou **DataInput**) também sejam comuns.

As seguintes informações devem ser exibidas nos comentários de documentação imediatamente antes da definição de uma interface:

- **Informar o objetivo.** Antes de outros desenvolvedores usarem uma interface, eles precisam saber qual é o conceito que ela encapsula. Em outras palavras, eles precisam saber qual é o seu objetivo. Um teste muito bom para saber se é preciso definir uma interface é identificar o grau de facilidade ou dificuldade com que você descreve sua finalidade. Se você tiver dificuldades para descrevê-la, provavelmente nem precisa da interface. Como o conceito de interfaces é novo na linguagem Java, as pessoas ainda

não têm experiência com o seu uso adequado e estão propensas a utilizarem as interfaces em excesso.

- **Como as interfaces devem e não devem ser usadas.** Os desenvolvedores precisam saber como uma interface deve ser usada e como ela não deve ser usada.

Pacotes

Há várias regras associadas à nomeação de pacotes. Por ordem, as regras são:

- **Identificadores são separados por pontos.** Para garantir a legibilidade dos nomes de pacote, a Sun sugere que os identificadores dos nomes de pacote sejam separados por pontos. Por exemplo, o nome de pacote java.awt é formado por dois identificadores: java e awt.
- **Os pacotes padrão de distribuição Java da Sun começam com o identificador "java".** A Sun reservou-se esse direito para que os pacotes padrão Java sejam nomeados com consistência, seja qual for o fornecedor do ambiente de desenvolvimento Java.
- **Os nomes de pacotes locais começam com um identificador cujas letras não são todas maiúsculas.** Os pacotes locais são usados internamente na organização e não serão distribuídos para outras organizações. Alguns exemplos desses nomes de pacote são persistence.mapping.relational e interface.screens.
- **Os nomes de pacotes globais começam com o nome de domínio da Internet para sua organização invertido.** Um pacote que será distribuído para várias organizações deve incluir o nome de domínio da organização original, com o tipo de domínio de nível superior em letras maiúsculas. Por exemplo, para distribuir os pacotes anteriores, eles devem ser nomeados como com.rational.www.persistence.mapping.relational e com.rational.www.interface.screens.

Você deve manter um ou mais documentos externos que descrevam a finalidade dos pacotes desenvolvidos pela sua organização. Para cada pacote, documente:

- **Os fundamentos do pacote.** Outros desenvolvedores precisam saber sobre o que se refere o pacote para que possam determinar se irão usá-lo e, caso ele seja um pacote compartilhado, se pretendem melhorá-lo ou ampliá-lo.
- **As classes contidas no pacote.** Inclua uma lista das classes e interfaces contidas no pacote com uma rápida descrição de uma linha para cada item, para que os outros desenvolvedores saibam o que o pacote contém.

B4.1.7. Tratamento de Erros e Exceções

Como filosofia geral, use exceções apenas para erros: erros de lógica e de programação, erros de configuração, dados corrompidos, esgotamento de recursos e assim por diante. Como regra geral, em condições normais e na ausência de sobrecarga ou falha de hardware, os sistemas não devem provocar exceções.

Use exceções para tratar erros de lógica e de programação, erros de configuração, dados corrompidos e esgotamento de recursos.

Diminua o número de exceções exportadas de uma determinada abstração.

Em sistemas de grande porte, o tratamento de um volume grande de exceções em cada nível dificulta a leitura e a manutenção do código. Às vezes, o processamento de exceções inibe o processamento normal.

Não use exceções para eventos freqüentes e previsíveis.

Não use exceções para implementar estruturas de controle.

Certifique-se de que os códigos de status tenham valores apropriados.

Ao usar código de status retornado por subprogramas como um parâmetro "externo", verifique se há um valor atribuído a esse parâmetro fazendo com que essa seja a primeira instrução executável no corpo do subprograma. Sistematicamente, coloque todos os status como sucesso ou como fracasso, por padrão. Pense em todas as possíveis saídas do subprograma, inclusive os controladores de exceção.

Realize verificações de segurança localmente. Não espere que o cliente o faça.

Se houver a possibilidade de um subprograma produzir resultados errados caso não seja informada determinada entrada, instale o código no subprograma para detectar e relatar entradas inválidas de maneira controlada. Não se baseie em comentários que solicitam que o cliente informe os valores apropriados. É praticamente garantido que mais cedo ou mais tarde o comentário será ignorado, resultando em erros de difícil depuração caso os parâmetros inválidos não sejam detectados.

B4.1.8. Diversos Padrões em questão

Reutilização

Se você adquirir ou reutilizar qualquer pacote ou biblioteca de classes Java de uma fonte externa, certifique-se de que seja 100% Java. Com a imposição desse padrão, você terá certeza de que o que está sendo reutilizado funcionará em todas as plataformas nas quais queira implantá-lo. É possível obter classes, pacotes ou applets Java de várias fontes, seja uma empresa de desenvolvimento de terceiros especializada em bibliotecas Java ou outro departamento ou equipe de projeto da organização.

Impostação de classes

A instrução de importação permite o uso de curingas para indicar os nomes das classes. Por exemplo, a instrução:

```
import java.awt.*;
```

Aciona todas as classes do pacote java.awt de uma vez. Na realidade, isso não é completamente verdadeiro. O que realmente ocorre é que todas as classes do pacote java.awt serão inseridas no seu código quando ele for compilado. As classes não utilizadas não serão inseridas. Embora pareça ser uma boa característica, ela diminui a legibilidade do código.

Otimização de códigos Java

A otimização do código é uma das últimas tarefas a serem consideradas pelos programadores, e não a primeira. Deixando a otimização para o fim, você otimizará apenas o código que precisar ser otimizado. Com freqüência, uma pequena porcentagem do código resulta em grande parte do tempo de processamento e esse é o código que você deverá otimizar. Um erro clássico cometido por programadores inexperientes é tentar otimizar todo o código, mesmo aquele que já é executado com bastante rapidez.

Não perca tempo otimizando códigos que não são importantes para ninguém!

Qual o objetivo da otimização do código? Os fatores mais importantes são carga fixa e desempenho em grandes entradas. O motivo disso é simples: a carga fixa domina a velocidade do tempo de execução para pequenas entradas, enquanto o algoritmo domina para grandes entradas. Segundo Koenig, um programa que funcione bem com pequenas e grandes entradas provavelmente funcionará bem com entradas de médio porte.

Os desenvolvedores que precisam criar software que funcione em várias plataformas de hardware e/ou sistemas operacionais devem estar cientes das idiossincrasias existentes em várias

plataformas. Operações que pareçam muito demoradas (como, por exemplo, a maneira como a memória e os buffers são manipulados) geralmente demonstram variações substanciais entre plataformas. É comum concluir que o código deve ser otimizado diferentemente para cada plataforma.

Outra questão a ser considerada durante a otimização do código é a prioridade dos usuários, porque as pessoas se incomodarão com certas demoras, dependendo do contexto. Por exemplo, os usuários provavelmente ficarão mais felizes com uma tela que apareça e, após oito segundos, carregue dados, em vez de uma tela que só apareça após cinco segundos para carregar os dados. Em outras palavras, a maioria dos usuários se dispõe a esperar um pouco mais, desde que eles obtenham algum feedback imediato — essa é uma informação importante na hora de otimizar o código.

Nem sempre você precisa fazer com que o código seja executado com mais rapidez para otimizá-lo na presença dos usuários.

Embora a otimização possa significar uma diferença entre o sucesso e o fracasso do aplicativo, nunca se esqueça de que é bem mais importante que o código funcione corretamente. Lembre-se de que um software lento mas que funcione é sempre preferível a um software rápido que não funcione.

Criação de teste em Java

Teste orientado a objetos é um tópico importante, que tem sido totalmente ignorado pelos responsáveis pelo desenvolvimento de objetos. A realidade é que você ou outra pessoa terá de testar o software criado, seja qual for a linguagem escolhida. Um equipamento de teste é o conjunto de funções de membro, algumas incorporadas nas próprias classes (denominadas testes internos) e outras em classes de teste especializadas usadas para testar o aplicativo.

Prefixe todos os nomes de funções de membro de teste com o termo "test". Dessa forma, você poderá encontrar rapidamente todas as funções de membro de teste no código. A vantagem dessa prefixação é facilitar a separação entre as funções de membro de teste e o código-fonte antes de compilar sua versão de produção.

Nomeie todos os métodos de teste forma consistente. O teste de método consiste em verificar se um único método é executado como definido. Todos os métodos do teste devem ser nomeadas de acordo com o formato "testMemberFunctionNameForTestName". Por exemplo, os métodos do equipamento de teste para experimentar withdrawFunds() devem incluir testWithdrawFundsForInsufficientFunds() e testWithdrawFundsForSmallWithdrawal(). Se você tem uma série de testes para withdrawFunds(), pode preferir criar um método denominado testWithdrawFunds(), que dispare todos eles.

Crie um único ponto para disparar os testes de uma classe. Desenvolva um método estático denominado testSelf() que dispare todas os métodos da classe.

Documente os métodos do equipamento de teste. Documente os métodos do equipamento de teste. A documentação deve incluir uma descrição do teste, assim como os resultados que se espera obter com ele.

B4.1.9. Referências

[AMB98] Ambler, S.W. (1998). *Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology*. Nova York: SIGS Books/Cambridge University Press.

[DES97] DeSoto, A. (1997). *Using the Beans Development Kit 1.0 February 1997: A Tutorial*. Sun Microsystems.

[GOS96] Gosling, J., Joy, B., Steele, G. (1996). *The Java Language Specification*. Reading, MA: Addison Wesley Longman Inc.

[KAN97] Kanerva, J. (1997). *The Java FAQ*. Reading, MA: Addison Wesley Longman Inc.

[LAF97] Laffra, C. (1997). *Advanced Java: Idioms, Pitfalls, Styles and Programming Tips*. Upper Saddle River, NJ: Prentice Hall Inc.

[LEA97] Lea, D. (1997). *Concurrent Programming in Java: Design Principles and Patterns*. Reading, MA: Addison Wesley Longman Inc.

[MEY88] Meyer, B. (1988). *Object-Oriented Software Construction*. Upper Saddle River, NJ: Prentice Hall Inc.

[NAG95] Nagler, J. (1995). *Coding Style and Good Computing Practices*. http://wizard.ucr.edu/~nagler/coding_style.html

[VIS96] Vision 2000 CCS Package and Application Team (1996). *Coding Standards for C, C++, and Java*. http://v2ma09.gsfc.nasa.gov/coding_standards.html